VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2019 Jan Bauer

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Jan Bauer**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Tieto Czech s.r.o.
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. RNDr. Petr Šaloun, Ph.D.**

Konzultant bakalářské práce: Mgr. Zdeněk Dřízga

Datum zadání: 01.09.2018

Datum odevzdání: 30.04.2019

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 4. dubna 2019 .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 4. dubna 2019

Tieto Czech s.r.o.
28. října 3346/91
702 00 Ostrava - Moravská Ostrava
IČO 64608051 DIČ CZ64608051

.....

Rád bych na tomto místě poděkoval společnosti Tieto Czech s.r.o., která mi poskytla možnost a dobré zázemí pro absolvování individuální odborné praxe. Zvláště děkuji Mgr. Zdeňku Dřízgovi za zadání a vedení projektu a také všem kolegům, kteří mi vždy ochotně poradili s daným problémem.
Dále bych chtěl poděkovat doc. RNDr. Petru Šalounovi, Ph.D. za vedení této práce a připomínky k jejímu vypracování.

Abstrakt

Cílem této bakalářské práce je popis mého působení a získaných zkušeností v rámci bakalářské praxe ve společnosti Tieto Czech s.r.o. (dále jen Tieto). Nejprve společnost Tieto představím a nastíním, čím se zabývá. Poté popíši, proč jsem si vybral možnost absolvování bakalářské praxe u této společnosti, dále moji roli a práci na přiděleném úkolu a postup jeho vypracování, technologie použité k implementaci a zhodnocení výsledku úkolu.

Klíčová slova: bakalářská praxe, odborná praxe, Tieto Czech s.r.o., Java, Spring framework, PotgreSQL, React.js, JPA, Hibernate

Abstract

An aim of this barchelor thesis is to describe my work a gained experiences within barchelor practise in company Tieto Czech s.r.o. (hereinafter reffered to as Tieto). First of all I introduce Tieto company and outline what it is doing. Then I describe why I have chosen to do barchelor practise in this company further my role and work on assigned task and process of its developing, technologies used to implementation and conclusion evaluating the result.

Key Words: bachelor thesis, professional practice, Tieto Czech s.r.o., Java, Spring framework, PotgreSQL, React.js, JPA, Hibernate

Obsah

Seznam použitých zkratek a symbolů

UML	–	Unified Modeling Language
LDAP	–	Lightweight Directory Access Protocol
HTTP	–	Hyper Text Transfer Protocol
REST	–	Representational state transfer
HTML	–	HyperText Markup Language
CSS	–	Cascading Style Sheets
JSON	–	Java Script Object Notation
JPA	–	Java Persistence API
DOM	–	Document Object Model
SQL	–	Structured Query Language
JPQL	–	Java Persistence Query Language
YAML	–	YAML Ain't Markup Language
DAO	–	Data Access Object
IDE	–	Integrated Development Environment
JVM	–	Java Virtual Machine
SDK	–	Software Development Kit
UI	–	User Interface
CRUD	–	Create, Read, Update, Delete
URL	–	Uniform Resource Locator
URI	–	Uniform Resource Identifier

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Tuto bakalářskou práci jsem si zvolil zpracovat formou odborné praxe ve společnosti Tieto Czech s.r.o., jelikož jsem si chtěl rozšířit znalosti nabyté při studiu a získat praktické zkušenosti s vývojem softwaru. Ve společnosti jsem pracoval na pozici softwarového vývojáře a popisuji práci na přiděleném projektu vyvíjené aplikace a nabyté zkušenosti v průběhu vývoje. Jedná se o webovou aplikaci na vytváření, přihlašování a správu firemních projektů, kdy mým úkolem byl návrh této aplikace z hlediska používání, architektury, použitých technologií a poté následná implementace. Chci se tedy zaměřit na postup vývoje této aplikace s danými problémy a jejich řešením, dále na využití technologií k jejímu vytvoření. V závěru práce uvádím znalosti nabyté při studiu, které jsem využil, dále jaké jsem získal a které mi v průběhu vypracovávání tohoto projektu scházely a nakonec zhodnotím dosažené výsledky.

2 Popis odborného zaměření firmy a pracovního zařazení

2.1 Společnost Tieto Czech s.r.o.

Tieto je největší skandinávská IT společnost se sídlem ve finském Espoo v Helsinkách. Společnost vznikla v roce 1968 s názvem Tietotehdas Oy, kdy v průběhu prvních let vyvíjela a spravovala IT systémy hlavně pro Union Bank of Finland, jeho zákazníky a lesnické společnosti. Od sedmdesátých let zákaznická základna společnosti rostla a poté v devadesátých letech společnost zaznamenala rapidní růst prostřednictvím několika akvizic, fúzí a strategických aliancí a v roce 1995 změnila název z Tietotehdas na TT Tieto, poté po spojení se švédskou společností Enator na TietoEnator a nakonec v roce 2008 na Tieto (což znamená finsky informace). Firma v dnešní době poskytuje komplexní služby v oblasti IT pro finanční, zdravotní, sociální, soukromý i veřejný sektor a služby pro vývoj produktů v oblasti komunikací a moderních technologií. V současnosti zaměstnává celkově přibližně 13 000 zaměstnanců ve více než 20 zemích s tím, že v České republice je to v Ostravě a v Brně více než 2600 lidí. Je proto jedním z největších zaměstnavatelů v Moravskoslezském kraji.[1]

2.2 Pracovní zařazení

Ve firmě jsem na stáži od března 2018 na pozici softwarového vývojáře. Mým úkolem je práce na zadaném projektu či na příležitostném úkolu, což je nejčastěji požadavek upravit či přidat určitou funkcionalitu v již existující aplikaci. Programuji webové aplikace, kdy nejčastěji na straně backendu se používá framework Spring boot v programovacím jazyce Java disponující REST API a na straně frontendu se používají React, Redux, Saga javascriptové knihovny. Jako databáze se používají podle potřeby jak SQL databáze, tak i NoSQL databáze.
Na projektu většinou pracujeme v týmu, kdy nám náš týmový manažer zadá požadavky na danou aplikaci a naším úkolem je návrh architektury této aplikace a následná implementace. Většina aplikací, které jsou vyvíjené stážisty, jsou určeny pro interní účely firmy.

3 Seznam úkolů zadaných studentovi v průběhu odborné praxe

3.1 Zadání projektu

Požadavkem bylo vytvořit webovou aplikaci pro zadávání, správu a přihlašování na nové firemní projekty. V tom smyslu, že hlavním záměrem aplikace je přidávání nápadů se specifikovanými technologiemi a popisem, kdy potom z těchto nápadů mohou vzejít projekty. Nápady mohou být přidány jakýmkoli uživatelem aplikace a potom schváleny nebo odmítnuty osobou s potřebnou autorizací - manažer, administrator, která může projekt také sama přímo specifikovat a přidat. Dále má uživatel možnost se na projekt ze všech zobrazených přidaných projektů přihlásit a napsat, proč by se chtěl na projektu podílet, kdy opět autorizovaná osoba tuto žádost posoudí a poté schválí nebo zamítne. Jinak je možné se z projektu také odhlásit. Dále je třeba zajistit potřebnou správu pro danou autorizovanou osobou, která může jak schvalovat jednotlivé žádosti uživatelů, tak také měnit stav projektů.

Aplikace by měla být schopna podle dat uživatele získaných z firemního LDAPu posílat upozornění pomocí e-mailů po provedení dané akce, např. schválení nápadu na projekt nebo změna stavu projektu či schválení/zamítnutí žádosti o přihlášení na projekt.

Aplikace bude také zabezpečena, tedy uživatel se do ní bude muset přihlásit pomocí firemního uživatelského jména a hesla a bude mít přidělenu danou roli.

Potom je třeba posoudit a zvolit použité technologie k implementaci této aplikace, kdy se jedná o perzistentní uložení dat v datové vrstvě, logiku práce s daty v doménové vrstvě a definovat vzhled zobrazení dat uživateli a jejich validace v prezentační vrstvě.

3.2 Řešení

Práce na projektu probíhala podle iterativní a inkrementální metodologie agilního vývoje softwaru SCRUM, kdy tedy bylo nutné na začátku definovat sprinty(iterace), za jakou dobu bude čeho dosaženo. Poté podle nich probíhaly průběžné prezentace části projektu s mým manažerem Mgr. Zdeňkem Dřízgou, kdy bylo třeba reagovat na nové připomínky a požadavky na změny a následně je tedy implementovat.

Na tomto projektu jsem celkově nakonec pracoval sám, tedy musel navrhnout dané řešení a poté ho i implementovat. Tento návrh a dané kroky jsem ovšem konzultoval s kolegy na příležitostných schůzkách.

4 Zvolený postup řešení zadaných úkolů

4.1 Návrh

4.1.1 Vize a funkční specifikace

Nejprve bylo třeba si definovat odpovědi na klíčové otázky práce s daty - tedy CO? JAK? KDE? KDO? KDY? Vycházel jsem ze zkušeností z předmětů Databázové a informační systémy a Vývoj informačních systémů, kdy bylo třeba definovat jednotlivé stavy entit a hlavně specifikovat, jak bude uživatel s aplikací pracovat a jak se bude chovat.

4.1.1.1 CO? Webová aplikace pro organizaci a správu firemních projektů.

4.1.1.2 JAK? Aplikace bude hlavně sloužit k přidávání nápadů na projekty, které budou posuzovány a poté schvalovány, kdy se z nich buď stanou projekty, anebo budou zamítnuty. Projekty bude také možno přidat, odebrat, změnit přímo osobou s danou rolí a následně bude možné se na tyto schválené projekty přihlásit či se z nich odhlásit, což bude také podřízeno následným schválením.

4.1.1.3 KDE? Aplikace bude využívána interně zaměstnanci ve firmě po přihlášení pomocí firemního uživatelského jména a hesla v kterémkoli webovém prohlížeči.

4.1.1.4 KDO? Role v aplikaci:
ZAMĚSTNANEC/UŽIVATEL (EMPLOYEE) – má základní možnosti

- přidání nápadu na projekt;
- zobrazení všech svých nápadů – ve 3 stavech;
- zobrazení všech projektů;
- zobrazení jeho přidělených projektů – ve 3 stavech;
- možnost přihlásit se na projekt;
- možnost odhlásit se z projektu.

MANAŽER (MANAGER) – má pravomoce jako ZAMĚSTNANEC, ale má možnost správy všech projektů a požadavků, kdy může:

- přidat projekt;
- odebrat projekt;
- upravit projekt;

- ukončit projekt;
- zobrazit všechny projekty;
- zobrazit všechny přidané nápady na schválení;
- zobrazit všechny žádosti o přihlášení na projekt;
- schválit/zamítnout nápad na projekt;
- schválit/zamítnout žádost o přidělení na daný projekt.

ADMINISTRÁTOR (ADMIN) – má pravomoce jako MANAŽER a jinak může přidělit/odebrat roli manažera danému uživateli

Role na projektu:

VEDOUCÍ (TEAMLEADER) – osoba/osoby zodpovědné za projekt, na které je možnost se obracet

ČLEN (MEMBER) – osoba/osoby přidělené k projektu

4.1.1.5 KDY? V případě, že uživatel bude mít nějaký nápad na projekt nebo bude chtít zobrazit schválené projekty s případným přihlášením se na daný projekt a v případě správy projektů a schvalování žádostí či změně role uživatele.

4.1.1.6 STAVY PROJEKTU (PROJECT STATE):

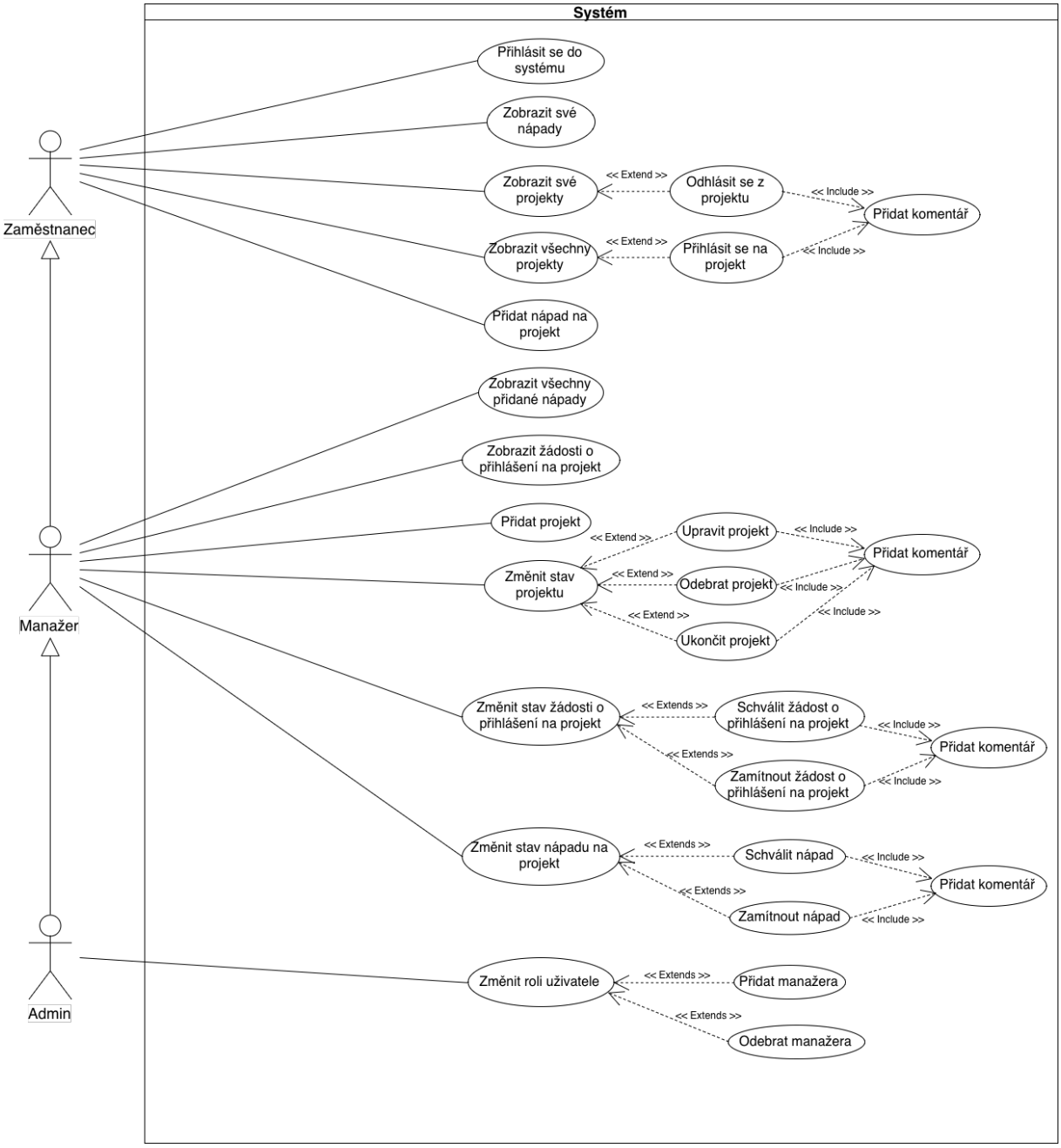
- ZPRACOVÁVÁN (IN PROCESS) – projekt, na kterém se pracuje
- DOKONČEN (FINISHED) – projekt, který je označen manažerem za dokončený

4.1.1.7 STAVY PŘIDĚLĚNÍ NA PROJEKT (ASSIGNMENT STATE):

- ŽÁDOST O SCHVÁLENÍ (JOIN REQUESTED) – stav přidělení po přihlášení uživatelem na daný projekt, který čeká na schválení manažerem
- SCHVÁLEN (APPROVED) – stav, kdy manažer schválí přihlášení na projekt a uživatel je přidělen k projektu s danou rolí teamleader/member
- ZAMÍTNUT (DECLINED) – stav, kdy manažer zamítne žádost o přihlášení na projekt

4.1.1.8 STAV NÁPADU (IDEA STATE):

- ČEKAJÍCÍ NA SCHVÁLENÍ (PENDING) – nápad, který byl přidán uživatelem a čeká na schválení od manažera
- SCHVÁLEN (APPROVED) – nápad, který byl manažerem schválen a stal se z něj projekt
- ZAMÍTNUT (DECLINED) – nápad, který byl manažerem zamítnut

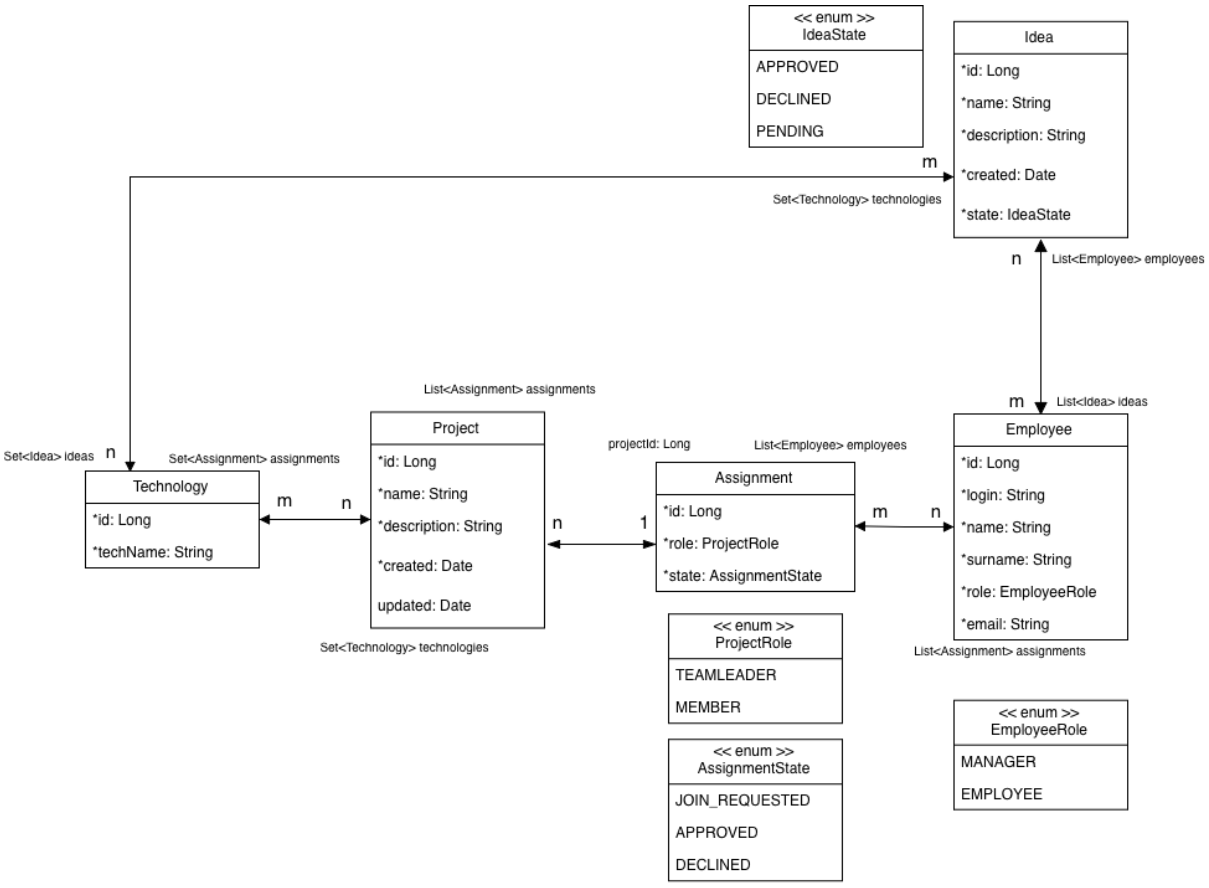


Obrázek 1: Use case diagram práce s aplikací jednotlivých aktérů s danými rolemi

4.1.2 Datový model

Poté bylo třeba sestavit datový model databáze pro ukládání dat aplikace, kdy bylo třeba posoudit, jaké vztahy mezi sebou jednotlivé entity budou mít a jaké atributy dat budou obsahovat.

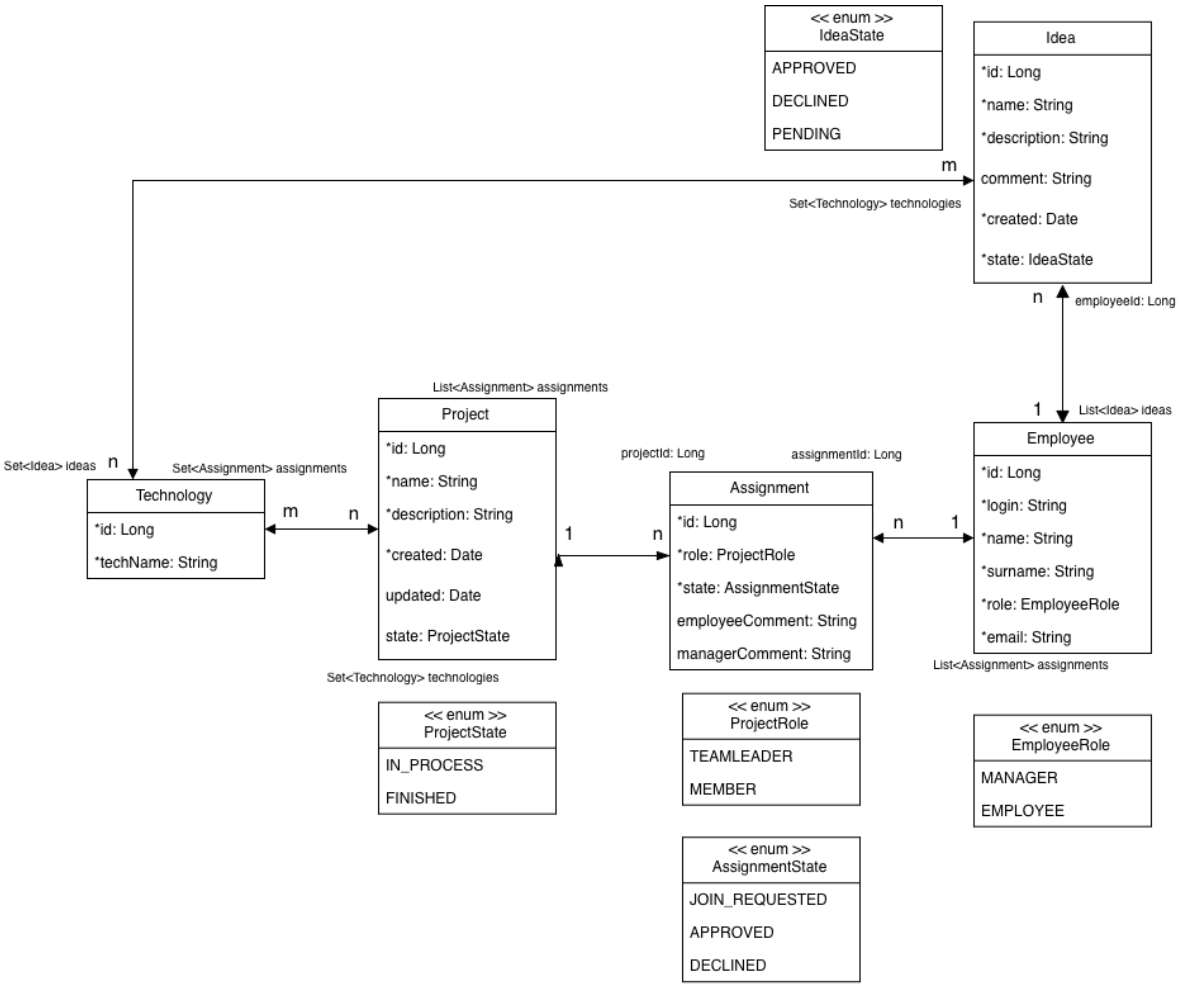
4.1.2.1 Prvotní návrh Neměl v sobě zahrnutou veškerou finální funkcionalitu .tzn neobsahoval ještě všechny stavy entit, a to stavy projektu, kdy jsem nepočítal s tím, že se projekty budou dát ukončit. Potom v něm nebyly zahrnuty komentáře přidávané k nápadům a k žádostem o přihlášení na projekt. Dále jsem navrhl toto řešení z hlediska toho, že nápady a projekty budou adresované jednotlivým osobám s rolí MANAŽER, který je následně bude posuzovat. Nakonec jsme tento návrh na jednotlivých iterativních schůzkách trochu přehodnotili.



Obrázek 2: JPA datový model aplikace první návrh

4.1.2.2 Současný návrh

Byly přidány stavy projektu, kdy je možné projekt označit uživatelem s rolí MANAGER za ukončený, byly přidány komentáře pro okomentování nápadu a žádosti o přihlášení na projekt a také byl dána jednotná správa dat a akcí všem uživatelů s rolí MANAGER, kterou jim může přiřadit uživatel s rolí ADMIN.



Obrázek 3: JPA datový model aplikace současný návrh

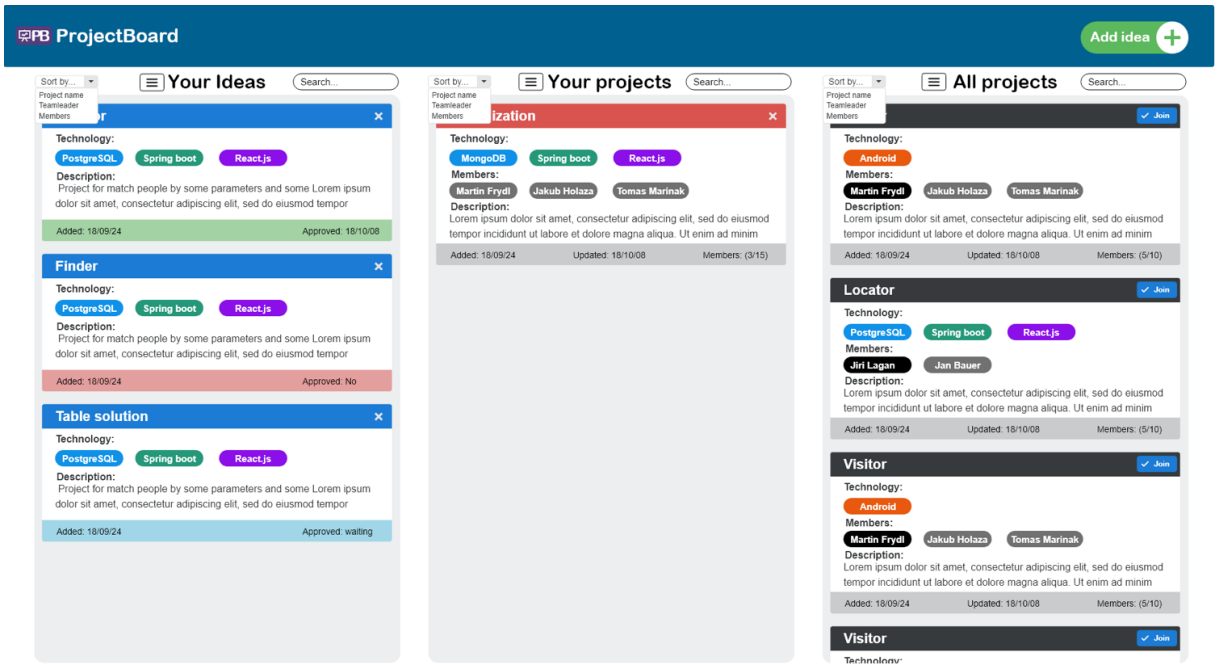
4.1.3 Návrh uživatelského rozhraní

Součástí návrhu byly předběžné návrhy uživatelského rozhraní, nebo-li wireframy či mock-upy, pomocí kterých bylo třeba definovat, jak asi bude aplikace vypadat, a jaké bude mít vizuální prvky. K tvorbě těchto wireframů jsem použil nástroj Adobe XD CC, na který jsme měli také na praxi workshop. Do tohoto nástroje je také možné stáhnout přímo Bootstrap CSS prvky, podle kterých je možno navrhnout rozhraní, které bude více věrné originálu, jelikož jsem Bootstrap pro tvorbu vizuálních komponent přímo použil.

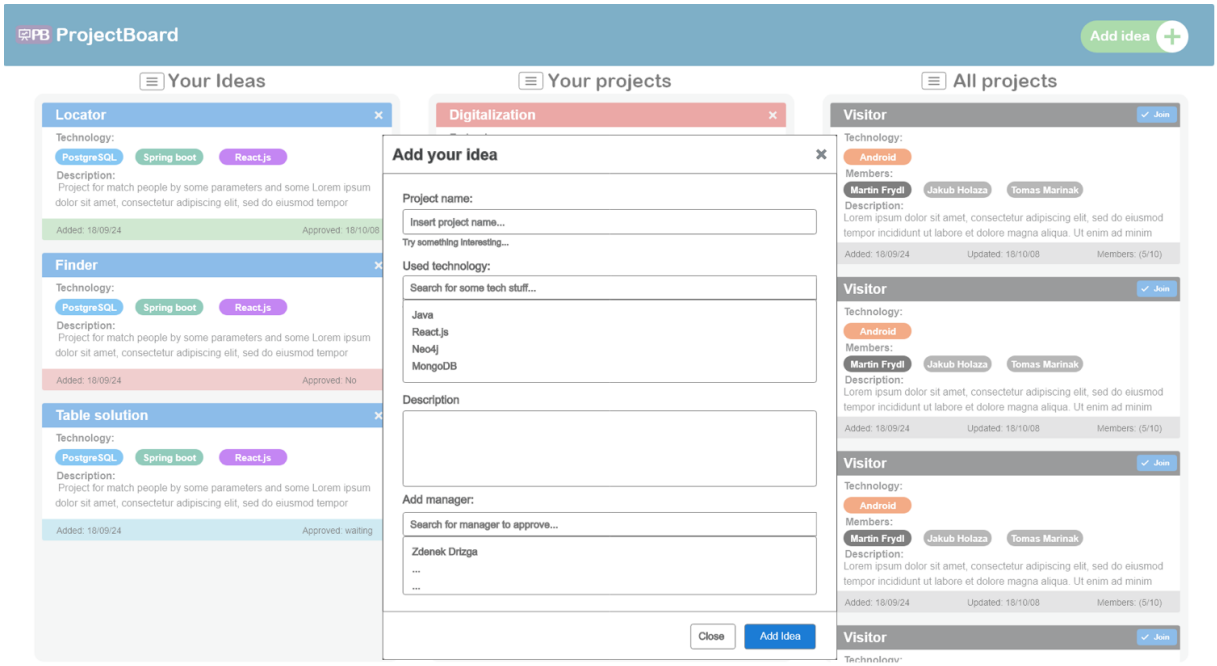
Rozhraní jsem rozdělil ze začátku do tří sloupců, tedy sloupec pro všechny uživatelovy nápady, sloupec pro všechny uživatelovy schválené projekty a sloupec se všemi projekty ze kterých

si uživatel může vybrat a přihlásit se na projekt pomocí tlačítka Join. V sloupcích jsou jednotlivé položky řešeny jako karty, které obsahují v záhlaví název projektu či nápadu a dále dané tlačítko. Dále tělo karty tvoří seznam použitých technologií, v případě projektu členové k němu přiřazení a popis toho čeho se daný nápad nebo projekt týká. Nakonec v zápatí jsou informace o vytvoření, změně či schválení dané položky. Potom v záhlaví stránky jsou k dispozici tlačítka, které vyvolají vyskakovací formulář, kde je v případě zaměstnance možno vyplnit parametry nápadu a v případě manažera pak parametry projektu a ten má také tlačítko s vyskakovací tabulkou pro schvalování požadavků na přihlášení na projekt.

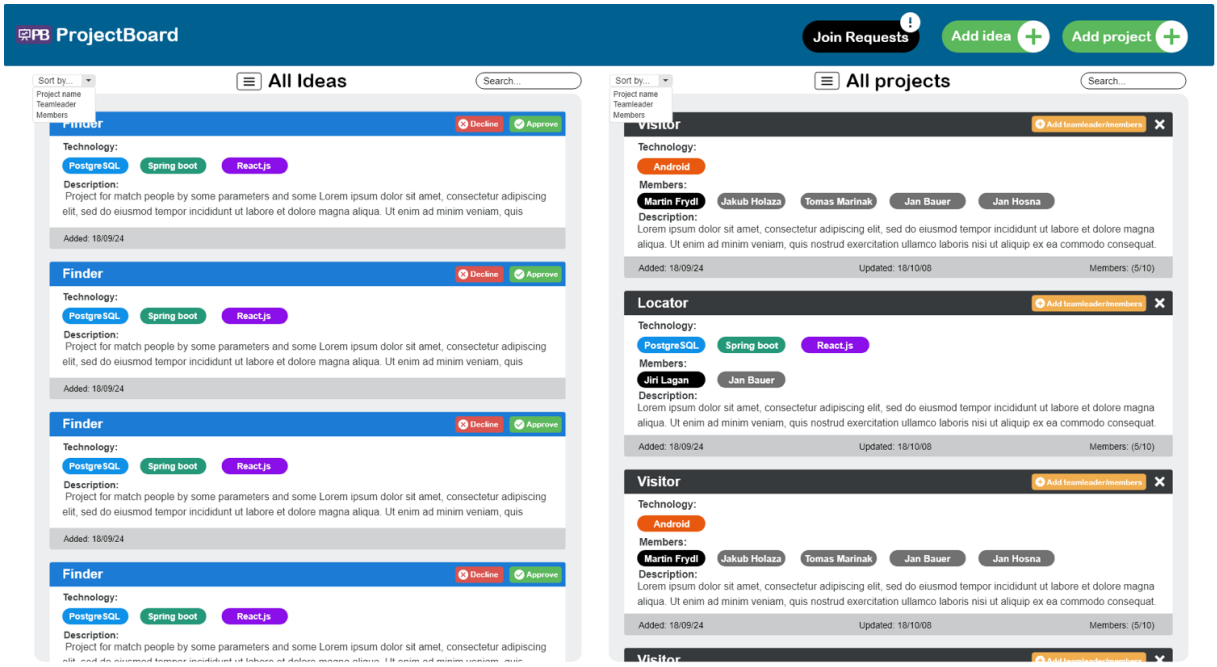
V průběhu vývoje a konzultací jednotlivých kroků bylo rozhraní postupně měněno, kdy bylo rozděleno do více stránek, dále byly změněny rozměry komponent, byly upraveny styly a barvy a další. Výsledný stav, funkcionalita a vzhled je popsán v závěru práce i s ukázkami.



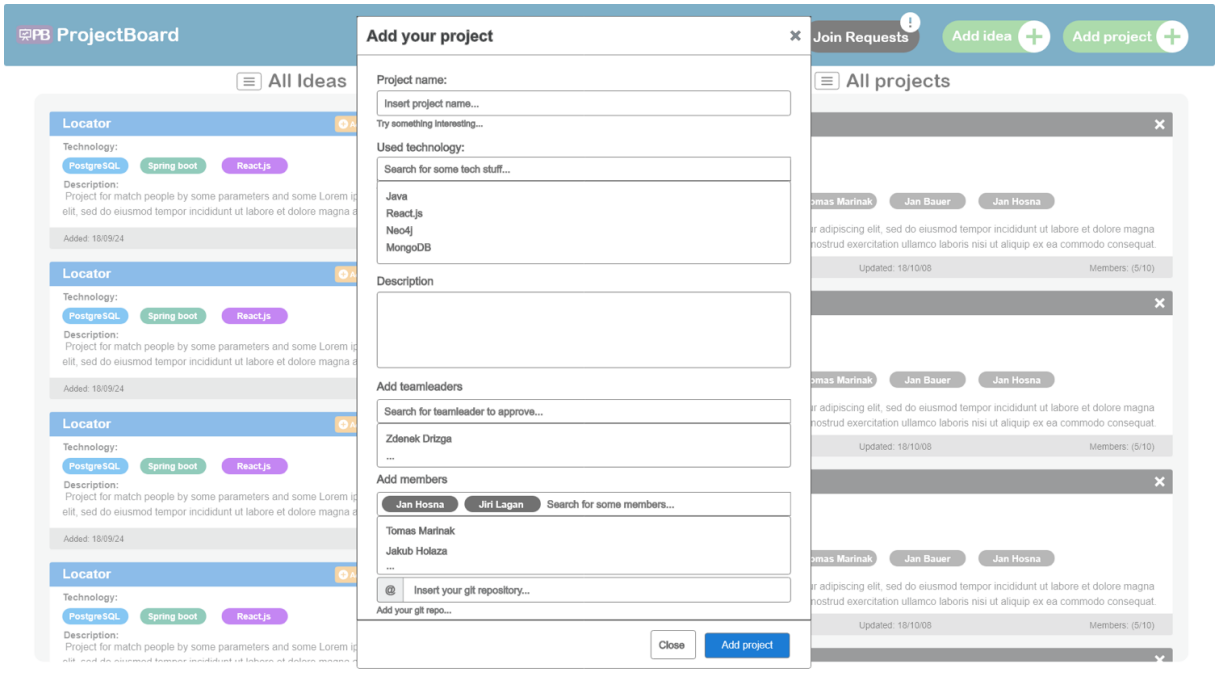
Obrázek 4: Employee page 1 – přehled



Obrázek 5: Employee page 2 – přidání nápadu



Obrázek 6: Manager page 1 – přehled



Obrázek 7: Manager page 2 – přidání projektu

4.2 Technická specifikace – popis využitých technologií a nástrojů

4.2.1 Datové uložiště

Jako datové uložiště bylo použito PostgreSQL, jakožto výkonný, open-source, objektově-relační databázový systém (ORDBMS) s více než třicetiletým vývojem, který si získal silnou reputaci spolehlivosti, robustnosti a výkonu. Je dostupný pod licencí MIT, jedná se tedy o volně dostupný a open-source software. Na jeho vývoji se podílí globální komunita vývojářů a firem. Z těchto důvodů a z důvodu, že byla potřeba relační databáze z hlediska datového modelu a také z hlediska předchozích zkušeností s touto databází to byla pro účely aplikace správná volba.[2]

4.2.2 Backend

Pro implementaci backendové části byl ve frameworcích použit programovací jazyk Java, jelikož je pro jejich vývoj určený. Je to tedy programovací jazyk a výpočetní platforma, která byla poprvé vydána společností Sun Microsystems v roce 1995. Java je rychlá, bezpečná a spolehlivá. Využívá se od notebooků po datacentra přes herní konzole až po vědecké superpočítače, mobilní telefony a na internetu. Napsaný kód jazyka je následně zkompilován do tzv. bajtového kódu a pak Java Virtual Machine (JVM), překládá tento bajtový kód do instrukcí nativního procesoru a umožňuje nepřímé provádění na úrovni operačního systému nebo platformy. Bytecode nelze spustit, pokud systém postrádá požadovaný JVM.

Vývoj programu Java vyžaduje sadu pro vývoj softwaru Java (SDK), která obvykle obsahuje kompilátor, interpret, generátor dokumentace a další nástroje používané k vytvoření úplné aplikace. [16]

4.2.2.1 Spring Framework Spring je framework od týmu Pivotal, který se používá pro backendovou část webových aplikací jako populární alternativa k Enterprise Java Beans (EJB) a pomáhá programátorům řešit myšlenku dependency-injection nebo to jak na sobě záleží nebo jsou propojeny jednotlivé moduly či části projektu. Dále pomáhá správně označovat a konfigurovat třídy a prvky kódu pro efektivní a přesné použití a také s dalšími aspekty jako plánováním a autentizací.

Pro zavedení Spring frameworku se používá Spring Boot, který navržen tak, aby také zjednodušil zavádění a vývoj nové Springové aplikace. Boot zaujímá konzistentní přístup ke konfiguraci a osvobozuje vývojáře od nutnosti definovat potřebnou opakující se konfiguraci boilerplate. V tomto směru se Boot snaží stát v čele v neustále se rozvíjejícím prostoru rychlého rozvoje aplikací.

Kvůli tomu, že platforma Spring IO byla kritizována za to, že má rozsáhlou XML konfiguraci s komplexním řízením závislostí se Spring Boot se snaží osvobodit vývojáře od potřeby XML a v daných případech od jednotvárnosti psaní import statementů.[4][3][5]

4.2.3 Frontend

Pro implementaci frontendové části byl použit programovací jazyk TypeScript, jelikož má pro vývoj rozsáhlejších JavaScriptových aplikací lepší přístup a výhody, jak je popsáno dále. Je to tedy open-source programovací jazyk vyvinutý a spravovaný společností Microsoft. Jedná se o přísnou syntaktickou nadmnožinu jazyka JavaScript a přidává do jazyka volitelné statické psaní a hlavně volitelné datové typy. TypeScript se zkompiluje do čistého, jednoduchého JavaScript kódu, který běží na libovolném prohlížeči na straně klienta i na straně serveru v Node.js, nebo v jakémkoliv JavaScript enginu, který podporuje ECMAScript 3 (nebo novější). [17]

4.2.3.1 React.js React je JavaScriptová knihovna pro vytváření znovupoužitelných UI komponent, tedy pro frontendovou část webové aplikace. Je vyvíjena a spravována společností Facebook a komunitou jednotlivých vývojářů a společností.

React nabízí jednodušší programovací model a lepší výkon díky tomu, že odstíní DOM a vytváří si vlastní virtuální DOM, který potom porovnává s tím skutečným v prohlížeči. Když najde rozdíly, tak ho co nejefektivnějším možným způsobem aktualizuje, takže není potřeba přesně definovat, které DOM elementy je třeba aktualizovat, jenom v komponentách deklarativně zadefinovat strukturu HTML skládáním JavaScriptových funkcí. React také může být vykreslován na serveru pomocí Node.js a může pohánět nativní aplikace pomocí React Native.

Jelikož React představuje "V"v pomyslném MVC, tedy pouze view - prezenční vrstvu aplikace, proto je třeba si vybrat a přidat další technologie, abychom získali kompletní sadu pro vývoj.[7]

4.2.3.2 JSX, HTML5, CSS3, SCSS, Bootstrap JSX(JavaScript XML) je rozšíření syntaxe jazyka JavaScript, které poskytuje způsob, jak vykreslit komponenty pomocí syntaxe HTML a CSS. SCSS (Sassy CSS) je nadmnožinou syntaxe CSS3. To znamená, že každá platná šablona CSS3 je platná i pro SCSS. V SCSS můžeme CSS kód zkrátit pomocí @mixin, takže nemusíme znovu a znovu psát vlastnosti jednotlivých atributů, ale můžeme to definovat pomocí funkce. Soubory SCSS používají příponu .scss.[8]

4.2.3.3 Redux.js Redux je předvídatelný stavový kontejner pro JavaScriptové aplikace. Pomahá psát aplikace, které se chovají důsledně a beží v různých prostředích (client, server, native) a jsou snadno testovatelné. Stará se o celý stav aplikace, který je uložen v stromovém objektu uvnitř jednoho tzv.store. Jediný způsob, jak změnit stavový strom, je vyvolat akci, tedy objekt popisující, co se stalo. K definování toho jak akce transformuje stavový strom, slouží tzv.reducery.[9]

4.2.3.4 Redux-Saga.js Redux-Saga.js je JavaScriptová knihovna, která se stará o vedlejší efekty aplikace jako asynchronní funkce práce s daty nebo přístup k mezipaměti prohlížeče. Snaží se o snadnější správu, efektivnější vykonávání, jednodušší testování a lepší manipulaci s chybami.[10][11]

4.2.4 Vývojové nástroje

4.2.4.1 Git Git je bezplatný a open-source distribuovaný verzovací systém, vytvořený Linusem Torvaldsem pro vývoj linuxového jádra, navržený tak, aby zvládl vše od malých po velmi velké projekty s rychlostí a efektivitou. Je určen pro správu kódu a stavu projektu, komunikaci a spolupráci na různých softwarových projektech a umožňuje vrátit se k předchozímu stavu nebo zobrazit celý jeho vývoj od doby, kdy byl projekt vytvořen. Osobně jsem se s ním setkal a nyní hlavně používám v podobě aplikace GitLab, který se na mé praxi používá a který poskytuje vše, co je potřeba ke správě, plánování, vytváření, ověřování, balení, uvolňování, konfiguraci, sledování a zabezpečení aplikací.[12][13]

4.2.4.2 Intellij Je vývojové prostředí (IDE) vyvíjené společností JetBrains primárně pro programovací jazyk Java, které jsem použil pro vývoj backendové části aplikace. Obsahuje funkce jako dokončení kódu analýzou kontextu, kódovou navigaci, která umožňuje přeskočit na třídu nebo deklaraci přímo v kódu, refactoring kódu a možnosti opravit nesrovnalosti prostřednictvím návrhů. Dále poskytuje integraci s nástroji pro tvorbu/balení, jako jsou například bower, gradle aj. Podporuje systémy pro správu verzí jako Git, apod. a je možnost se přímo připojit k databázi

a pracovat s ní. Také umožňuje přidávat pluginy pomocí kterých lze do IDE přidat další funkce. V současné době je k dispozici okolo 1600+ pluginů.[15]

4.2.4.3 Webstorm Je další vývojové prostředí (IDE) od společnosti JetBrains určené pro vývoj webových aplikací a jiných JavaScriptových aplikací, které disponuje víceméně jinak stejnými funkcemi jako IDE Intellij a které jsem použil pro vývoj frontendové části aplikace.[15]

4.2.4.4 DataGrip Je databázové vývojové prostředí (IDE) také od firmy JetBrains, které jsem použil hlavně pro práci s databázemi, kdy jsem přistupoval k lokální a testovací databázi.[15]

4.2.4.5 Chrome debug tools pro React a Redux Hojně jsem používal debug nástoje přímo v prohlížeči Chrome, které poskytují velké množství funkcí pro kontrolu toho co a jak se ve frontendové části projektu děje, kdy jsem hlavně používal testování uprav HTML a CSS syntaxe, konzoli aplikace a přehled sítě požadavků na REST API, které v aplikaci probíhali. Dále jsem využil dostupných zásuvných debug nástrojů pro React a Redux z Chrome web store.

4.2.5 Další technologie

4.2.5.1 Mail service Služba, která byla implementována dříve jedním z kolegů a je možné ji získat pomocí Maven dependency. Umožňuje definovat strukturu emailu přímo v kódu aplikace a přistupovat k ní přes dependenci injection a poté je třeba vytvořit a zadat šablonu emailu, definovanou v .html a v .txt, která musí být uložena v daném repozitáři na GitLabu, do níž poté služba vloží definované proměnné části označené ve složených závorkách {{ }}. Používá se k tomu JinJava, což je šablonový Java engine založený na syntaxi šablony django, přizpůsobený k tomu, aby poskytl jinja šablony.

4.2.5.2 Zabezpečení – IDP (Intrusion Detection and Prevention) service Na závěr bylo nutné aplikaci zabezpečit, tak že uživatel se musí před začátkem vždy přihlásit a jeho data musí zůstat v bezpečí. K tomuto účelu nabízí Spring Boot službu Spring security, která poskytuje aplikaci jak autentizaci, tak i autorizaci, kdy může být rozšířena o vlastní požadavky na rozšíření, například ve spojení s frameworkem OAuth 2.0 a rozdělit naši aplikaci do více modulů jako Auth, kde se řeší zabezpečení a Core, kde se řeší logika aplikace.

Díky tomu, že na firemním serveru je nasazená IDP služba, která přejímá roli OAuth modulu a tedy chrání business logiku před širokou škálou útoků a podezřelých aktivit např. SQL injekce, DoS apod. a řeší správu access a refresh tokenů, tak bylo možné pouze definovat Spring security konfiguraci a v YAML konfiguračním souboru aplikace specifikovat přístup Spring security k této IDP službě, pak jsou přístupové údaje autentizovány touto službou a autentizovaný uživatel dostane access token a je poté zapsán do SecurityContextHolderu aplikace a může pracovat s REST API.

```yaml
security:
  basic:
    enabled: false
  oauth2:
    client:
      client-id: ...
      client-secret: ...
      access-token-uri: https://.../idp/oauth/token
      user-authorization-uri: https://.../idp/oauth/authorize
    resource:
      token-info-uri: https://.../idp/oauth/check_token
```

Výpis 1: YAML konfigurace zabezpečení

4.3 Vypracování/Implementace a architektura aplikace

Po vypracování a schválení návrhu popsaného výše byla na řadě samotná implementace aplikace. Nejprve bylo třeba začít od datové vrstvy, poté přejít na doménovou vrstvu a vytvořit business logiku práce s daty a nakonec vytvořit prezentační vrstvu, která data získá z doménové vrstvy a zobrazí je v dostatečně přívětivé formě.

Pro datovou vrstvu bylo třeba vytvořit PostgreSQL databázi popsanou výše, kdy jsem tuto databázi získal z webových stránek vývojářů a poté pomocí nástroje PgAdmin 4 nadefinoval její parametry, tedy hlavně název, uživatel, heslo a port pro lokální použití.

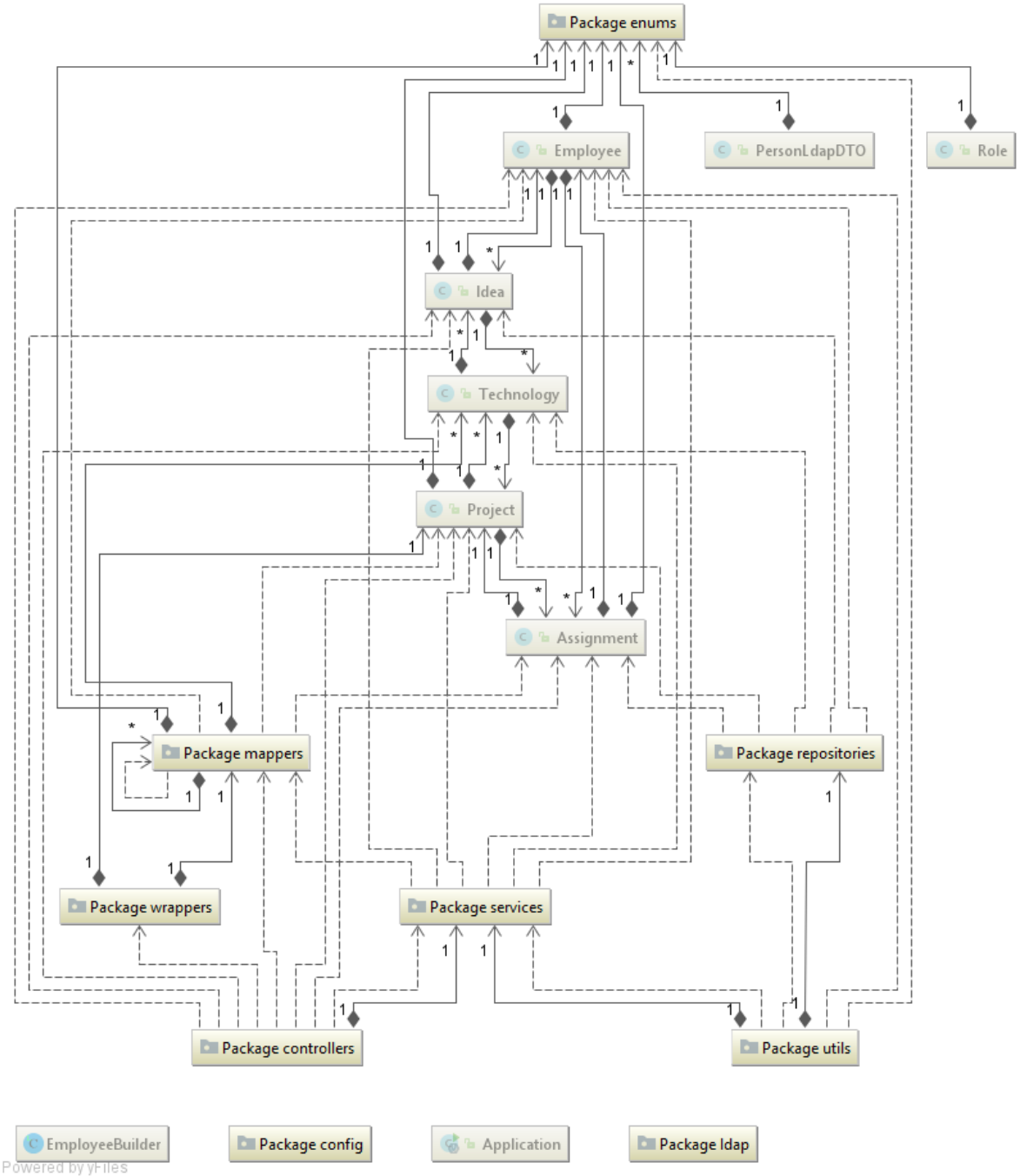
Následně jsem vytvořil backendový projekt, tak že jsem nejprve na webových stránkách Spring bootu nadefinoval základní nástavení Spring aplikace přes Spring Initializer s potřebnými závislostmi a importoval jsem ho ve vývojovém prostředí Intellij přes Maven pom.xml. Pak jsem vytvořil konfigurační soubor YAML a definoval jsem přístupové údaje připojení k dříve vytvořené databázi. Poté jsem začal vytvářet základní balíčkovou strukturu tedy Modely(Models), DAO Repozitáře(Repositories), Služby(Services) a REST ovladače(Controllers). Následně jsem vytvořil třídy pro všechny modely podle navrženého datového modelu s danými vzájemnými vztahy a použitím technologie JPA/Hibernate, kdy jsem musel zjišťovat z dokumentací a internetových zdrojů, jaké jsou nejlepší přístupy pro vytváření a mapování datového modelu pomocí těchto technologií. Tyto záležitosti jsem probíral se svými kolegy. Po vytvoření backendového projektu jsem otestoval, zda při spuštění projektu se databáze správně inicializuje. Po několika pokusech a změnách byla struktura databáze ve správném stavu. Dále bylo pro každý model třeba v balíčku repozitářů definovat jejich DAO třídu, která implementuje rozhraní JpaRepository, které zajišťuje práci s těmito modely namapovanými v databázi. Pak jsem vytvořil třídy služeb pro každý model, kde jsem začal přidávat logiku práce s modely, což zabralo asi nejvíce času. Ve službách bylo využito dependency injection pro práci s repozitáři a v daných službách

byla také podle potřeby využita e-mailová služba, kdy v příslušných metodách byla přímo definována struktura emailu. V závislosti na e-mailové službě bylo třeba přidat i třídu s plánovači pro posílání e-mailů v daných intervalech definovaných pomocí tzv. Cron výrazů např. "0 0 12 * * ?"znamená, že se metoda provede každý den v poledne. Následně jsem vytvořil třídy ovladačů pro každý model a definoval jednotlivé metody URL koncových bodů, které se odkazují na třídy služeb, pro přístup z prezentační vrstvy. Poté jsem začal s testováním této vrstvy pomocí nástroje Postman, kde jsem zasílal jednotlivé HTTP požadavky na dříve definované metody ovladačů.

Následně jsem vytvořil frontendový projekt v React.js, kdy jsem nejprve pomocí balíčkového manažera npm vytvořil React aplikaci se základní konfigurací ve vývojovém prostředí Webstorm. Poté jsem do aplikace přidal balíčky Redux.js, který se stará o správu dat a Saga.js, která se stará o asynchronní funkce a přistupování k REST API koncovým bodům na backendu. Dále jsem přidal definované konfigurační soubory, které se používají ve frontendových aplikacích ve firmě a jednotlivé vizuální Bootstrap komponenty definované pro použití v React aplikaci. Poté bylo třeba vytvořit strukturu modelů stejnou jako v backendové části a zároveň k nim vytvořit mapovací funkce, které mapují modely, které jsou zasílány v JSON objektech z backendu na Immutable.js objekty tzv. Mapy. Pak bylo třeba definovat jednotlivé zobrazení podle role uživatele, kdy tedy po každý typ bylo třeba vytvořit daný kontejner, který obsahuje soubory constants.ts, actions.ts, reducer.ts, index.tsx, sagas.ts a rules.ts. V souboru constants.ts jsem definoval potřebné konstanty, pomocí kterých je celá aplikace propojená skrze akce a reducer klíče, pod kterými jsou jednotlivá data uložena v Redux reduceru. V souboru actions.ts jsem musel definovat akce s danými konstantami, které se spouští nad reducerem a sagou a nesou potřebné údaje. V souboru index.tsx jsem začal vytvářet celkový vzhled stránky za pomocí dříve definovaných Bootstrap komponent, které využívají akce pro vykonávání funkcí daných vizuálních prvků a zobrazují data, která jsou uložena v reduceru. V reducer.ts jsem musel definovat jednotlivé reducery, které jsou vázané na dané konstanty a obsahují potřebnou částečnou logiku, hlavně tedy mapování JSON objektů na Immulatable.js objekty pomocí mapovacích funkcích jednotlivých modelů a ukládají je pod danými klíči do jednotného statu aplikace. Dále jsem definoval jednotlivé Saga funkce, které jsou vázané na příslušné konstanty a zadal jsem potřebné stejné URL definované v backendových ovladačích pro zasílání HTTP požadavků. V souboru rules.ts jsem ještě definoval validační pravidla pro dané vstupy za pomoci balíčku Validate.js a regex výrazů. Poté jsem testoval propojení frontendové a backendové části a na závěr po úspěšném propojení jsem musel přidat konfiguraci zabezpečení aplikace pomocí Spring security, které bylo využito u koncových bodů, kde kontrolovalo, zda je daný uživatel autentizován a byly také definovány volně přístupné koncové body a pomocí IDP služby, která byla konfigurována v YAML souboru, pro autentizaci tokenů posílaných v hlavičkách jednotlivých HTTP požadavků.

Na úplný závěr se po připomínkách při testování opravovali dané chyby.

4.3.1 Struktura doménové/backendové části projektu



Obrázek 8: Struktura backendového projektu

4.3.1.1 Modely

```
@NoArgsConstructor
@AllArgsConstructor
@ToString
@EqualsAndHashCode
@Getter
@Setter
@Entity
@Table(name = "EMPLOYEE")
public class Employee implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NonNull
    @Size(min = 8, max = 8, message = "Login must be 8 characters long")
    private String login;

    @NonNull
    private String name;

    @NonNull
    private String surname;

    @Enumerated(EnumType.STRING)
    private EmployeeRole role;

    @NonNull
    @Email(message = "Email is not in the right format")
    private String email;

    private byte[] photo;

    @OneToMany(cascade = {CascadeType.PERSIST, CascadeType.MERGE}, fetch =
        FetchType.LAZY, mappedBy = "employee")
    private List<Assignment> assignments;
```

```
    @OneToMany(cascade = {CascadeType.PERSIST, CascadeType.MERGE}, fetch =
        FetchType.LAZY, mappedBy = "employee")
    private List<Idea> ideas;
}
```

Výpis 2: Employee model

Příklad modelu, kdy se jedná o model zaměstnance (Employee), který má své atributy od generovaného id až po svou roli a potom má své vazby na další modely tedy na modely Ideas a Assignments. V modelu jsou použity JPA anotace pro definování struktury modelu a jeho vztahů a pak tedy díky anotaci @Entity je tento model podle jeho atributů a vztahů v rámci objektově-relačního mapování namapován do databáze pomocí frameworku Hibernate, který implementuje JPA.[6]

Použité JPA anotace:

- @Entity – povinná anotace, která definuje, že třída může být namapována na tabulku, tedy že se v databázi vytvoří tabulka s názvem třídy. Zapotřebí, aby třída mohla být namapována, ovšem je, aby třída obsahovala proměnnou s anotací @Id. Další proměnné ve třídě budou namapovány odvozením jako sloupce v tabulce, pokud je třeba definovat nějaké další podmínky pro sloupce tak nad těmito proměnnými můžeme použít anotaci @Column s danými parametry.
- @Table – volitelná anotace, která explicitně určí název tabulky namapované entity a poté například při vyváření dotazů v nativním SQL v repozitářích se na tabulku dotazuje pod tímto definovaným jménem.
- @Id @GeneratedValue(strategy = GenerationType.IDENTITY) – povinná anotace, která označuje proměnnou jako primární klíč této entity v databázi. Anotace za ní s ní spojená konfiguruje způsob inkrementování tohoto primárního klíče, kdy jsou typy strategií generování této hodnoty a v mém případě mém jsem nakonec místo strategie AUTO, která vybere automaticky nejlepší způsob generování podle typu databáze, zvolil raději přímo strategii IDENTITY, která toto generování nechává na databázi a umožňuje jí generovat novou hodnotu s každou operací vložení, což je z hlediska databáze je to velmi efektivní a také kvůli tomu, aby bylo možné přidávat řádky s generovaným id do tabulky i normálně pomocí SQL a ne jen pomocí Hibernate.
- @Size – volitelná anotace, která definuje omezení jakou velikost by proměnná typu String měla mít.

- @Enumerated(EnumType.STRING) – v tomto případě povinná anotace, jelikož proměnná je typu Enum tímto deklarujeme, že jeho hodnota by v tomto případě měla být efektivně převedena na typ String
- @Email – volitelná anotace, která slouží k validaci toho zda proměnná obsahuje platnou formu emailu.
- @OneToMany – definuje vazbu s jinými modely, což znamená, že jeden řádek v tabulce je mapován na více řádků v jiné tabulce, kdy je povinný atribut mappedBy, který označuje název proměnné tohoto modelu v mapovaném model s nímž je vytvořena vazba, dále je zde označeno atributem FetchType, který říká, že při dotazování se na model nebudou zároveň vybrány i všechny řádky mapovaného modelu a pak, že díky atributu CascadeType bude možné řídit změny v tomto případě z hlediska ukládání a změny v těchto mapovaných modelech.

Dále model obsahuje anotace pluginu Lombok, které mu v tomto případě slouží k jeho vytvoření a práci s ním, tak že mu zajišťují typy konstruktorů a getterů a setterů, které nemusíme přímo definovat v kódu, ale Lombok je za nás doplní.

Použité Lombok anotace:

- @NoArgsConstructor – vygeneruje konstruktor bez argumentů.
- @AllArgsConstructor – vygeneruje konstruktor s 1 parametrem pro každou proměnnou. ve třídě. Pole označená @NonNull znamenají null kontrolu těchto parametrů.
- @ToString – vygeneruje přetíženou metodu toString pro tuto třídu.
- @EqualsAndHashCode – vygeneruje metody Equals a HashCode.
- @Getter – vygeneruje gettery pro každou proměnou.
- @Setter – vygeneruje settery pro každou proměnnou.
- @NonNull – znamená kontrolu toho zda pole není null, tedy daná proměnná nesmí být null.

4.3.1.2 Repository

```
@Repository
public interface ProjectRepository extends JpaRepository<Project, Long> {

    Project findByName(String name);

    @Query("select distinct p from Project p join p.assignments a join a.
        employee e where e.id = ?1")
    List<Project> getProjectsByEmployeeId(Long id);

    @Query("select distinct p from Project p join p.assignments a join a.
        employee e where e.id <> ?1 and p.projectState = 'IN_PROCESS'")
    List<Project> getProjectsWithoutEmployeeById(Long id);

    @Query("select distinct p from Project p LEFT JOIN p.technologies t LEFT
        JOIN p.assignments a LEFT JOIN a.employee e WHERE (t.techName IN :
        technologies OR :technologies IS NULL) " +
            "AND (e.login IN :employees OR :employees IS NULL) AND (LOWER(p.name
                ) LIKE :nameStr OR :nameStr is null)")
    List<Project> getFilteredProjects(@Param("nameStr") String nameStr,@Param("
        employees") List<String> employees,@Param("technologies") List<String>
        technologies);
}
```

Výpis 3: Project repozitář

Na příkladu vidíme repozitář modelu Project, který impementuje JpaRepository, což nám umožní práci s databází jejíž přístupové údaje jsou definované v konfiguračním souboru YAML, tedy k účelu správy relačních dat v databázi JPA framework vygeneruje potřebné základní CRUD operace nad databází, které můžeme použít a potom je ještě možné vytvářet vlastní CRUD operace buď v dotazovacím jazyce JPQL, který je zaměřený více objektově, kdy respektuje nadefinované vztahy, které mezi sebou entity mají a dotaz je vykonáván nad objekty aplikace (entitami) v projektu tedy odstiňuje programátora od specifik konkrétního datového úložiště. Jinak lze také použít i přímo nativně dotazovací jazyk SQL. V tomto případě jsem použil JPQL(Java Persistence Query Language) a potřeboval jsem definovat dotazy na projekt v závislosti na daném zaměstnanci, pak hlavně dotaz pro filtrování projektů podle názvu, členů a technologii.[6]

Použité Spring anotace:

- @Repository – označí třídu jako Bean a tedy potom když Spring provádí skenování komponentů, tak tuto třídu vloží do application contextu a poté je přístupná skrze dependency

injection. Je to, ale speciální typ anotace @Component pro označení třídy typu DAO, která slouží pro práci s databází, kdy také umožňuje překlad DAO výjímek na Spring vyjímky.

- @Query – hodnota této anotace je buď JPQL nebo SQL dotaz prováděný nad databází.
- @Param – označení parametru funkce, který se s daným názvem použije v dotazu s anotací @Query.

```yaml
spring:
  datasource:
    url: jdbc:postgresql://localhost:5432/...
    username: postgres
    password:
    initialization-mode: always
  jpa:
    properties:
      hibernate:
        jdbc:
          lob:
            non_contextual_creation: true
    generate-ddl: true
    hibernate:
      ddl-auto: update
```

Výpis 4: YAML konfigurace připojení na databázi

4.3.1.3 Service

```java
public interface IdeaService {
    List<Idea> getAll();

    Idea getById(Long id);

    Idea getByName(String name);

    Idea saveOrUpdate(Idea idea);

    void delete(Idea idea);

    boolean deleteOneIdea(Long ideaId);
}
```

Výpis 5: Idea služba rozhraní

```java
@Service
public class IdeaServiceImpl implements IdeaService {

    private IdeaRepository ideaRepository;
    private TechnologyRepository technologyRepository;

    @Autowired
    public IdeaServiceImpl(IdeaRepository ideaRepository, TechnologyRepository
        technologyRepository){
        this.ideaRepository = ideaRepository;
        this.technologyRepository = technologyRepository;
    }

    @Override
    public List<Idea> getAll() {
        return this.ideaRepository.findAll();
    }

    @Override
    public Idea getById(Long id) {
        return this.ideaRepository.findById(id).orElse(null);
    }

    @Override
    public Idea getByName(String name) {
        return this.ideaRepository.findByName(name);
    }

    @Override
    public Idea saveOrUpdate(Idea idea) {
        this.ideaRepository.save(idea);
        return idea;
    }

    @Override
    public void delete(Idea idea) {
        idea.getTechnologies().forEach(tech -> {
            tech.getIdeas().remove(idea);
```

```
        });
        idea.getEmployee().getIdeas().remove(idea);
        this.ideaRepository.delete(idea);
    }

    @Override
    public boolean deleteOneIdea(Long ideaId) {
        Idea idea = this.getById(ideaId);
        if(idea == null)
            return false;
        this.delete(idea);
        return true;
    }
  ...
}
```

Výpis 6: Idea služba implementace rozhraní

Na příkladu vidíme službu modelu Idea, tedy Service layer, kde je nejprve interface s deklarovanými metodami a pak jeho implementace, kde jsou tyto metody definovány. V těchto metodách probíhá hlavní logika aplikace, kdy každý model má svou službu, která se stará o práci a provádění změn s tímto modelem za pomocí repozitářů.

Použité Spring anotace:

- @Service – stejně jako anotace @Repository je @Service speciálním typem anotace @Component a tedy označí třídu jako Bean a potom je k ní možné přistupovat pomocí dependency injection. Toto označení slouží pro třídy které obsahují business logiku, tedy jsou v service layer.
- @Autowired – je anotace ve Spring frameworku používaná pro řízení dependency injection tzn. umožní použít jinou třídu (nebo tedy Beanu) a její služby v této dané třídě, pokud byli třídy označeny některým typem anotace @Component a tím byli přidány do dependency injection kontejneru, tedy ApplicationContextu při skenovaní komponent Springem nebo byli do tohoto skenování přidány pomocí anotace @ComponentScan

4.3.1.4 Controller

```
@RestController
@RequestMapping("/assignment")
public class AssignmentController {

    private AssignmentService assignmentService;

    @Autowired
    public AssignmentController(AssignmentService assignmentService){
        this.assignmentService = assignmentService;
    }

    @PreAuthorize("isAuthenticated()")
    @PostMapping("/employee/{employeeId}/project/{projectId}")
    public HttpEntity<?> createAssignment(@PathVariable("employeeId") Long
        employeeId, @PathVariable("projectId") Long projectId, @RequestBody
        Assignment assignment){
        if(this.assignmentService.create(employeeId,projectId,assignment)){
            return new ResponseEntity<>(HttpStatus.CREATED);
        }
        return new ResponseEntity<>(HttpStatus.BAD_REQUEST);
    }
  ...
}
```

Výpis 7: Assignment ovladač

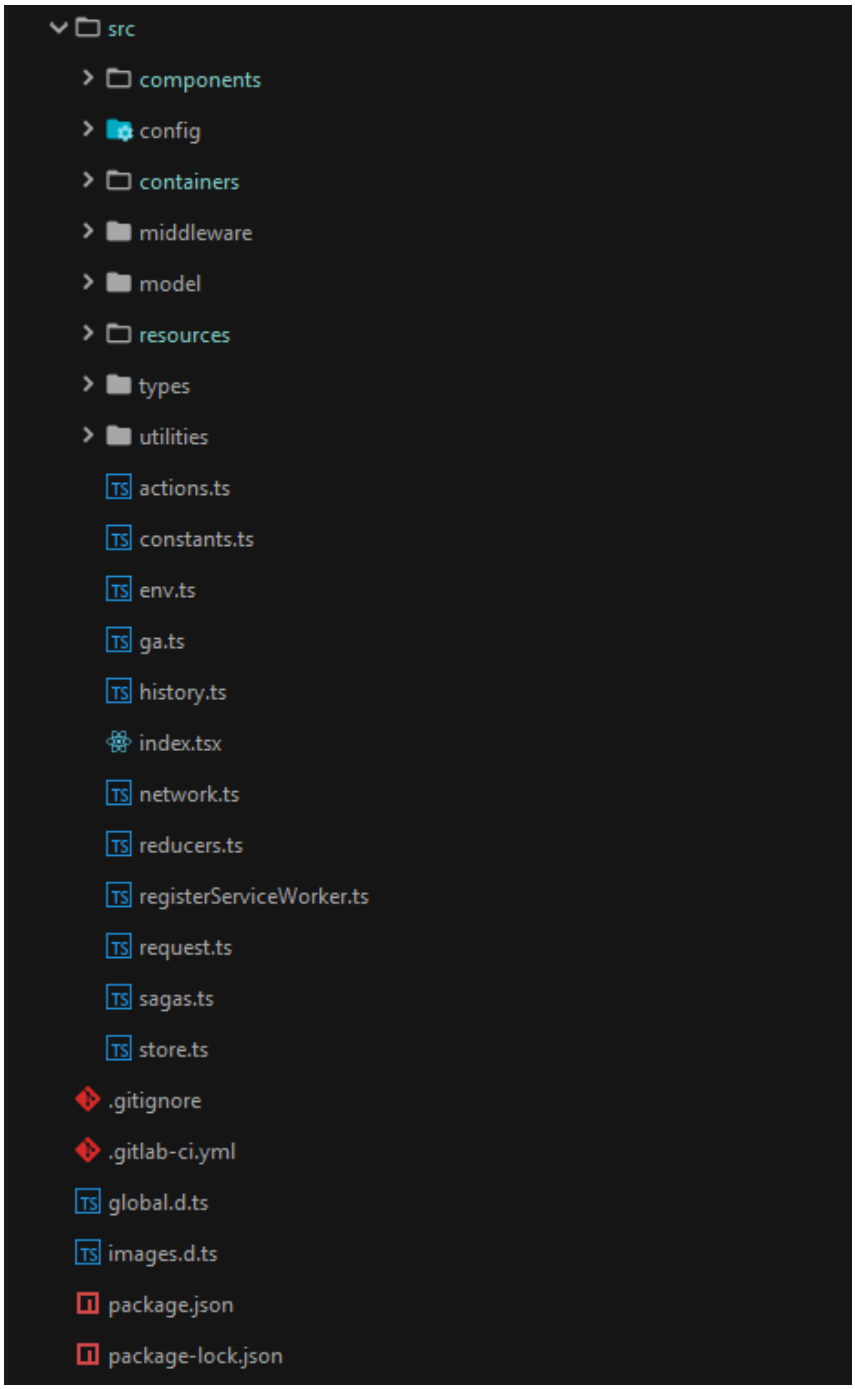
Na příkladu vidíme Assignment ovladač, který se stará o koncové body REST API – anotace @RestController nad tímto modelem, kdy je možno na ně přistupovat z frontendu, ale tyto přístupy musí být autentizované, kdy se o kontrolu stará Spring security anotace @PreAuthorize("isAuthenticated()“). Ovladače potom volají dané služby, které jim buď vrátí potřebné data nebo potvrzení, že daná akce proběhla v pořádku a podle toho zasílají zpět objekty, které jsou serializované do JSON objektů a nebo HTTP statusy o výsledku akce z HTTP požadavku.

Použité Spring anotace:

- @RestController – je speciálním typem anotace @Controller, která je zase speciálním typem anotace @Component, která umožnuje práci s třídou v rámci dependenci injection, jak bylo popsáno výše a kromě anotace @RequestMapping obsahuje i anotaci @RequestBody.
- @RequestMapping – umožňuje mapovat URL řetězce na dané třídy nebo metody.

- @PathVariable – slouží pro mapování zástupného řetězce z hierarchické části URI na danou proměnou v parametru metody.
- @RequestParam – slouží pro mapování zástupného řetězce z dotazové části URI na danou proměnou v parametru metody.
- @RequestBody – umožňuje automaticky deserializovat JSON objekty z HTTP požadavků na objekty daný tříd v Javě.
- @GetMapping, @PostMapping, @PutMapping, @DeleteMapping – vestavěné anotace pro pro zpracování různých typů příchozích HTTP požadavků, tedy GET, POST, PUT a DELETE.
- @PreAuthorize – anotace Spring security, která slouží pro kontrolu, zda o přístup žádá v tomto případě autentizovaná osoba.

4.3.2 Struktura prezentační/frontendové části projektu



Obrázek 9: Struktura frontendového projektu

4.3.2.1 Component Vizuální prvky aplikace, znovupoužitelné části kódu, které jsou v tomto případě tedy ve velké míře Bootstrap komponenty definované pro použití v Reactu s několika vlastními úpravami, kdy je pro každou komponentu zajištěna potřebná funkčnost s danými parametry a potom se použije v jakémkoli kontejneru, kde se jí dají dané údaje a potom jsou skrze ni zobrazeny.

```
interface CardProps {
  cardContainerClass?: string;
  ...
  height?: string;
}

class Card extends React.PureComponent<CardProps> {
  private renderCardHeader() {
    const { headerTitle, headerClass, headerContent } = this.props;
    return headerTitle ? (
      <div className={`card-header ${headerClass}`}>
        <span className="font-weight-bold">{headerTitle}</span>
        {headerContent || null}
      </div>
    ) : null;
  }

 ...

  private renderCardFooter() {
    const { footerContent, footerClass } = this.props;
    return footerContent ? (
      <div className={`card-footer ${footerClass}`}>{footerContent}</div>
    ) : null;
  }

  public render() {
    const { cardContainerClass, bodyClass } = this.props;
    return (
      <div className={`card ${cardContainerClass}`} style={{height: `${this.
          props.height}`, overflow: "hidden"}}>
        {this.renderCardHeader()}
        <div className={`card-body ${bodyClass}`} style={{ padding: "1.1em" }}
            >
```

```
          {this.renderCardTitle()}
          {this.renderCardSubtitle()}
          {this.renderCardText()}
          {this.renderCardContent()}
        </div>
        {this.renderCardFooter()}
      </div>
    );
  }
}

export default Card;
```

Výpis 8: Card komponenta

4.3.2.2 Container Starají se o konkrétní část aplikace a jsou většinou na nějaké trase (Route) poskytované node modulem react-router. Obsahují veškeré nutné prvky, které jsou potřeba, tedy získávání dat z backendu, ukládání dat do frontendového stavu(state) aplikace, validace vstupů a zobrazení dat. Veškeré změny a práce s daty jsou prováděny nebo spíše propojeny skrze konstanty a vyvolány skrze akce definované ve vlastních souborech. Skládají se vlastně ze tří použitých knihoven popsaných výše. Pro zobrazení je využito Reactu a jeho komponent v index.tsx, pro uložení dat v aplikaci slouží Redux v reducers.ts a pro asynchronní akce slouží Saga v sagas.ts.

Dělí se na:

- actions – Akce vyvolávané v aplikaci nejčastěji v index.tsx, které mohou nést potřebná data, které jsou potom využitelné v reduceru nebo v saze nebo jenom spustí potřebné zpracování dat. Odesílají se pomocí metody dispatch().

```
export const fetchEmployee = (id: number) => action(FETCH_EMPLOYEE,id);

export const fetchAllEmployees = () => action(FETCH_ALL_EMPLOYEES,[]);

export const addIdea = (employee: EmployeeType, idea: IdeaType, technology
    : any[]) => action(ADD_IDEA,{employee,idea,technology});
```

Výpis 9: Employee kontejner akce

- constants – Konstanty akcí, díky kterým jsou propojeny všechny části aplikace, které akce ovlivňují. Jsou definovány pro každý kontejner a název je dán podle účelu dané vyvolané

akce např. získání dat přes REST API nebo uložení dat z uživatelova vstupu ve formuláři. Dále soubor obsahuje definované klíče pro reducer pod kterými jsou dané data uloženy a pomocí, kterých je k nim možné později přistupovat.

```
Employee constants
export const CLEAR = "EmployeeContainer/CLEAR";
export const FETCH_EMPLOYEE = "EmployeeContainer/FETCH_EMPLOYEE";
...
export const FILTER_PROJECTS = "EmployeeContainer/FILTER_PROJECTS";

export const cReducer = {
    employeeData: "employeeData",
    allEmployeesData: "allEmployeesData",
    allProjectsData: "allProjectsData",
   ...
    joinOnProjectComment: "joinOnProjectComment",
    error: "error",
    validationErrors: "validationErrors",
};
```

Výpis 10: Employee kontejner konstanty

- index – Zde je definován vzhled celého kontejneru v jedné třídě, kdy jsou využity komponenty a TSX. Jednotlivé prvky jsou definované ve funkcích a poté pospojované v jiných funkcích a výsledek je poté vrácen v metodě render(), která zobrazí výsledek nadefinovaného kontejneru. Dále jsou zde využívány akce nejčastěji v jednotlivých komponentách a k datům aplikace se přistupuje pomocí odkazu na reducer. Všechny akce a reducery všech kontejnerů jsou propojeny pomocí funkce connect(), u které je třeba definovat metodu zvanou mapStateToProps(), která umožní propojit a používat reducery skrze dané rekvizity či argumenty (props) a potom je třeba definovat metodu zvanou mapDispatchToProps(), která umožňuje propojit a používat akce zase skrze předané parametry.

```
interface EmployeeContainerProps {
  employeeContainerReducer: EmployeeContainerReducerInterface;
  ...
  autoActions: typeof autoActions;
}

class EmployeeContainer extends React.Component<EmployeeContainerProps> {
  public state = {
    color: "#e0e0e0",
```

```
    ...
    filterOn: false,
  };

  public componentWillMount() {
    const id: number = this.props.loginContainerReducer.getIn([
      cLogin.accountTypeData,
      cEmployee.id
    ]);
    this.props.actions.fetchEmployee(id);
    this.props.actions.fetchEmployeeIdeas(id);
    this.props.actions.fetchEmployeeProjects(id);
    this.props.actions.fetchAllProjects(id);
  }

  private handleChange = (key: any, value: any) => {
    this.props.actions.handleAddIdeaModalChange(key, value);
  };
...
 private renderMain() {
    return (
      <React.Fragment>
         {this.renderSidebar()}
         {this.state.mainPageShow
             ? this.renderAllProjectsMainPage()
              : this.renderMyDesktop()}
      </React.Fragment>
    );
  }

  public render() {
    return (
      <React.Fragment>
        {this.renderMain()}
      <Footer style={{bottom: "0px", position: "absolute", width: "100%"}}>
          <img height={"30px"} src={TietoLogo} alt="footer-tieto-logo" />
        </Footer>
      </React.Fragment>
    );
```

```
  }
}

const mapStateToProps = (state: StoreState) => ({
  employeeContainerReducer: state.EmployeeContainerReducer,
  autoCompleteContainerReducer: state.AutoCompleteContainerReducer,
  managerContainerReducer: state.ManagerContainerReducer,
  loginContainerReducer: state.LoginContainerReducer
});

const mapDispatchToProps = (dispatch: Dispatch) => ({
  actions: bindActionCreators(actions, dispatch),
  loginActions: bindActionCreators(loginActions, dispatch),
  autoActions: bindActionCreators(autoActions, dispatch)
});

export default connect(
  mapStateToProps,
  mapDispatchToProps
)(EmployeeContainer);
```

Výpis 11: Employee kontejner index.tsx

- reducers – Poskytují jednotné uložiště dat aplikace. Jsou definované pro každý kontejner a určují, jak se stav dat aplikace mění v reakci na odesílané akce. Obsahují nejprve deklaraci základní struktury typů dat, které jsou v něm uloženy, pak deklaraci typu reduceru, jako Immutable mapy, dle dříve deklarovaných typů a dalších parametrů. Dále definici základního stavu uložiště, který se použije při inicializaci nebo při vyčištění všech dat po zavolání dané akce. Nakonec následuje definice změn celkového stavu uložiště pomocí akcí a daných konstant, kdy po každé akci tento stav zůstane v korektním stavu a potom také záleží zda daná akce spustila také požadavek na backend a podle dané odpovědi dle HTTP kódu (FULFILLED, REJECTED) jsou provedeny dané příkazy.[9]

```
export interface EmployeeContainerStateProps {
  employeeData: EmployeeType;
  employeeIdeas: IdeaType;
  employeeProjects: ProjectObjectType;
  allProjectsData: ProjectObjectType;
  employeeIdeaRecordCreator: IdeaType;
}
```

```
export type EmployeeContainerActionsType = ActionType<typeof actions>;

export type EmployeeContainerReducerInterface = ImmutableMap<
  EmployeeContainerStateProps,
  string,
  any
>;

const initialState: EmployeeContainerReducerInterface = Map({
  [c.allEmployeesData]: [],
  ...
  [c.employeeRecordCreator]: Map().setIn([cIdea.name],"").setIn([cIdea.
      description],"").setIn([c.joinOnProjectComment],"").setIn([
      cCommentObject.employeeComment],""),
  [c.filterString]: "",
});

export default (
  state = initialState,
  action: EmployeeContainerActionsType
): EmployeeContainerReducerInterface => {
  switch (action.type) {
    case CLEAR:
      return initialState.setIn([c.filterString], "");
    case `${FETCH_EMPLOYEE_IDEAS}`:
         return state.setIn([c.employeeIdeasLoading], true);
    case `${FETCH_EMPLOYEE_IDEAS}${FULFILLED}`:
      const ideas = action.data as any;
      const employeeIdeas: IdeaType[] = ideas.map(IdeaMapper);
      return state
        .setIn([c.employeeIdeas], employeeIdeas)
        .setIn([c.employeeIdeasLoading], false);
   case `${FETCH_EMPLOYEE_IDEAS}${REJECTED}`:
      return state
        .setIn([c.error], action.data)
        .setIn([c.employeeIdeasLoading], false);
...
    default:
```

```
      return state;
  }
};
```

Výpis 12: Employee kontejner reducer

- rules – Poskytují validaci vstupů s pomocí knihovny validate.js, kdy je možné použít vestavěné funkce, ale i například regex. Využil jsem je v kontejnerech pro zaměstnance, ve formuláři pro zadání parametrů nápadu, pro přidání komentáře a v kontejneru pro manažera ve formuláři pro zadání parametrů nového projektu nebo při jeho editaci.

```
export const EmployeeFieldsRules = {
    [cIdea.name]: {
        presence: {
            allowEmpty: false,
        },
      length: {
        minimum: 1,
        maximum: 25,
        message: "must be at least 1 and less than 25 characters long"
      }
    },
    ...
};
```

Výpis 13: Employee kontejner pravidla

- sagas – Poskytují asynchronní funkcionalitu a jsou spouštěny pomocí konstant v akcích, ze kterých si můžou vzít data a sestavit a poslat požadavek s těmito daty na daný endpoint backendu, či nějaké data z backendu získat a poté je předat v akci do reduceru pro uložení.[10][11]

```
export function* fetchEmployee(action: EmployeeContainerActionsType) {
  const id: number = action.data as any;
  try {
    const response = yield (call as any)(
      request,
      url(`employee/${id}`),
      get({})
    );
    yield put(fulfilled(FETCH_EMPLOYEE, response));
```

```
  } catch (e) {
    yield put(rejected(FETCH_EMPLOYEE));
  }
}
...

export default function* sagas() {
  yield all([
    yield takeLatest(FETCH_EMPLOYEE, fetchEmployee),
    ...
  ]);
}
```

Výpis 14: Employee kontejner saga

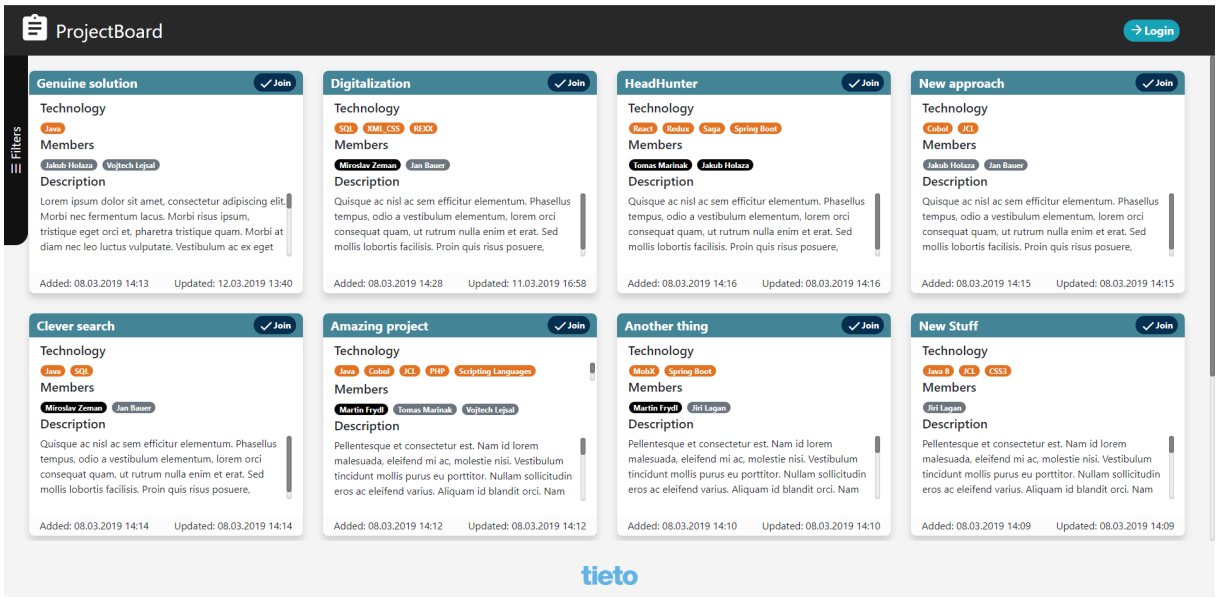
5 Výsledná aplikace

Ve výsledné verzi bylo postupně provedeno několik zásadních změn oproti původnímu návrhu, kdy hlavně rozhraní bylo rozděleno do více stránek a do velké míry už není situováno do karet, jak tomu bylo na začátku.

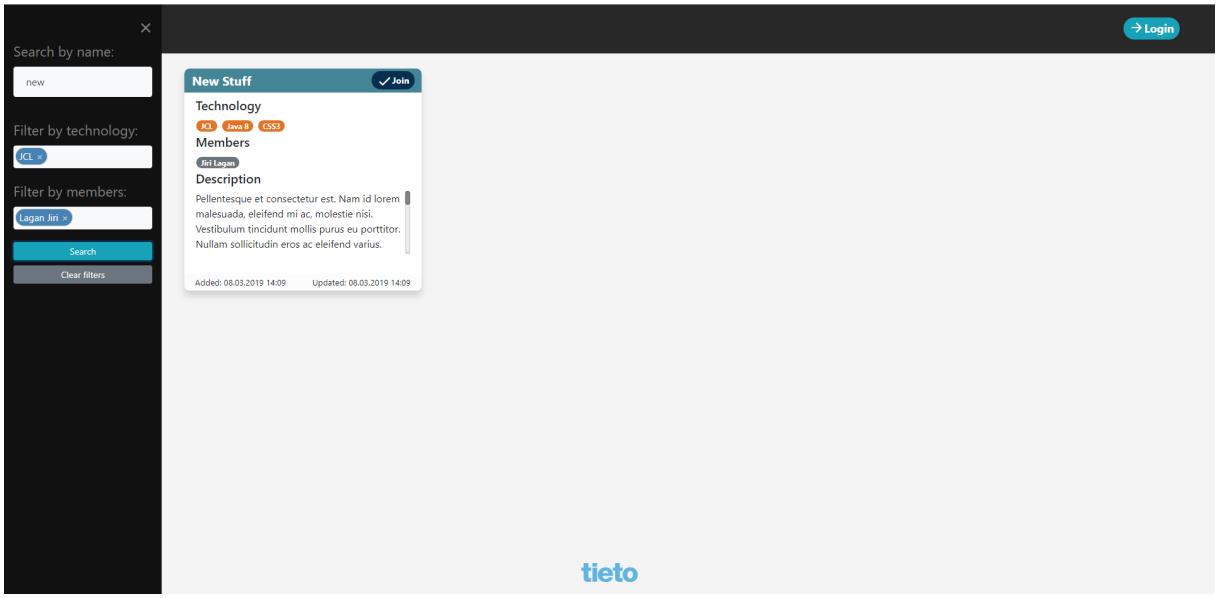
Kdy tedy byla přidána tzv. landing page před samotným přihlášením jako úvodní stránka aplikace, kde se zobrazí všechny projekty v jednotlivých menších kartách a je možné je filtrovat s pomocí výsuvné komponenty podle jména, technologií a členů, které je možné přidat pomocí našeptávacích vyhledávacích polí nebo je možné kliknout na danou technologii či člena na kartě projektu, což tuto možnost také přidá do vyhledávání. Následně pokud se uživatel chce přihlásit na daný projekt nebo přidat nápad, tak se musí přihlásit pomocí vyskakovacího formuláře kliknutím na tlačítko Login v záhlaví nebo Join na kartě projektu, kdy po přihlášení se v záhlaví zobrazí tlačítko pro vyskakovací formulář pro zadání parametrů nápadu a zobrazí se mu stejné rozhraní s filtry jako na landing page, ale pouze s projekty, na které není přihlášen nebo mu nejsou zamítnuty. Pak má k dispozici stránku My desktop s dvěma sloupci, kde jsou jeho nápady a přiřazené projekty v podobné stylu jako v návrhu, ale do sloupců byly přidány karty, kde jsou rozděleny jednotlivé stavy položek, kdy projekty jsou rozděleny podle toho, zda byly schválené, čekají na schválení, nebyly schválené, uživatelem odhlášené projekty a dokončené projekty. V případě nápadů je toto rozdělení do karet podle schválených, čekajících na schválení a neschválených.

V případě, že uživatel je současně manažer, tak byla přidána položka do záhlaví navigace s touto stránkou, která je také rozdělena na sloupce, kde první je rozdělen do karet. První karta obsahuje seznam se všemi projekty, které je možné upravit, označit za dokončené či je odstranit pomocí daných tlačítek. Pak karta se seznamem dokončených projektů s možností odstranění toho projektu a karta se seznamem všech přidaných nápadů s možností je schválit či neschválit pomocí tlačítek, které vyvolají vyskakovací formulář pro přidání komentáře a v případě schválení, pak ještě přidaní členů na nově vzniklý projekt z tohoto nápadu. Dále je zde k dispozici sloupec s detaily dané položky ze seznamu, kdy při kliknutí na danou položku se vyrendrují její parametry. Nakonec je zde sloupec s tabulkou všech žádostí o přihlášení na projekt, kde každá položka obsahuje tlačítka, kdy je nejdříve možné zobrazit komentář žádajícího uživatele a pak buď zvolit možnost potvrzení nebo odmítnutí pomocí daného tlačítka, které zase vyvolá vyskakovací formulář pro přidání komentáře pro žádajícího.

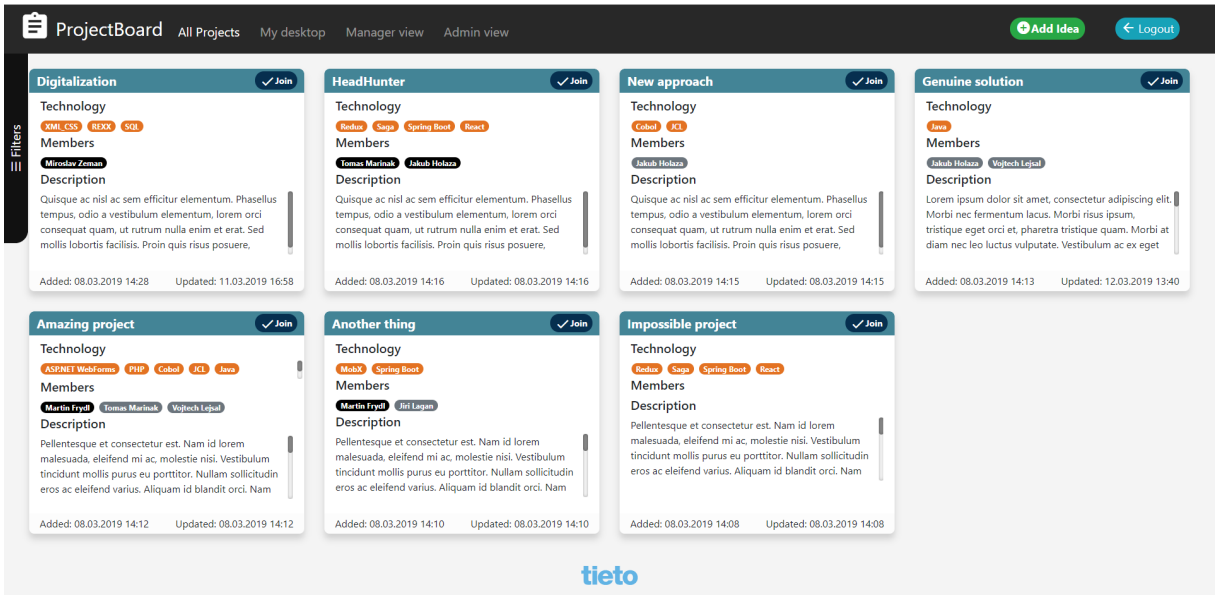
Pokud má uživatel současně roli administrátora, tak se mu v záhlaví navigace zobrazí položka s administrační stránkou. Tato stránka slouží pro správu manažerských rolí v aplikaci. Na stránce se nachází tabulka se všemi uživateli s rolí manažer a pomocí našeptávacího vyhledávacího pole je možné přidat nového manažera ze seznamu všech uživatelů dříve přihlášených do aplikace. Dále je možné některému uživateli z tabulky odebrat tuto roli pomocí daného tlačítka.



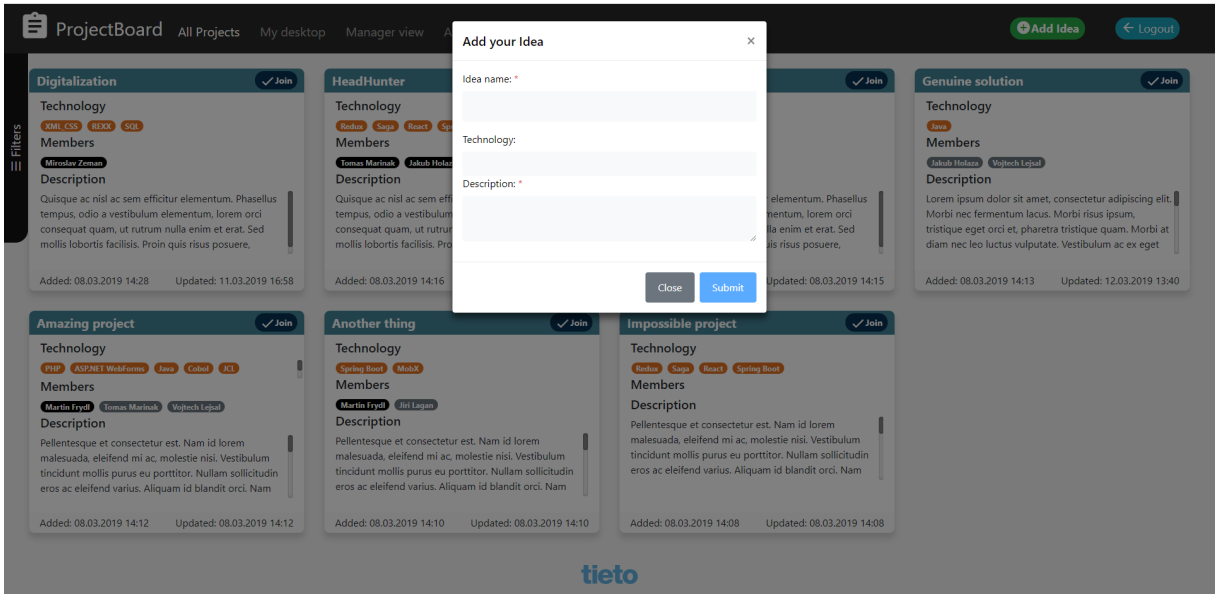
Obrázek 10: Landing page 1



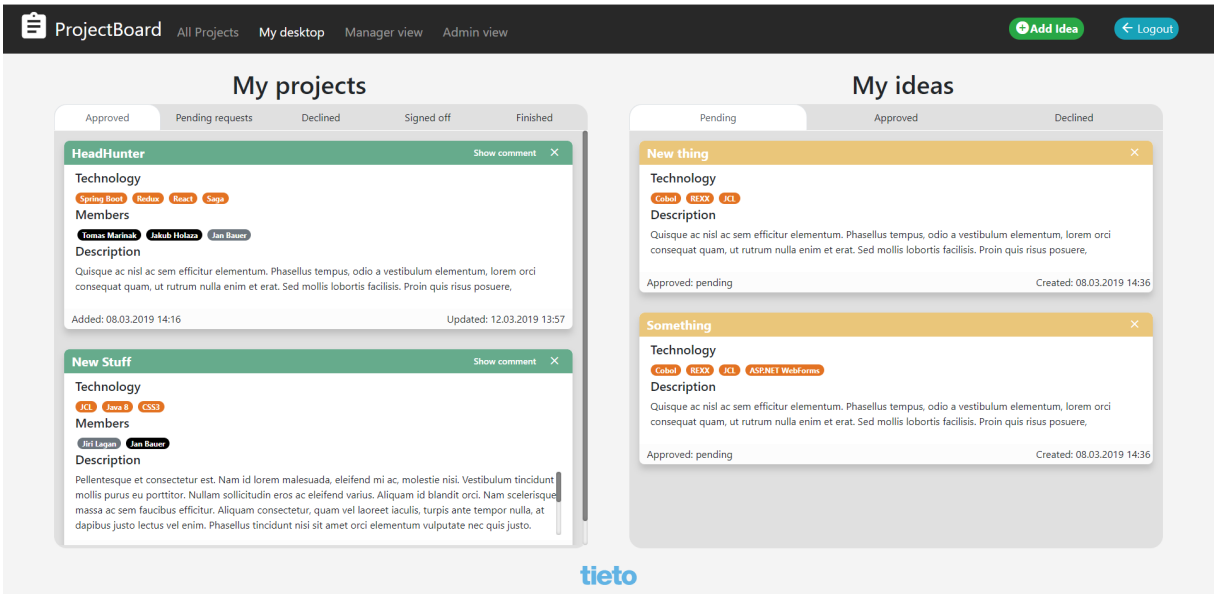
Obrázek 11: Landing page 2 – filtry



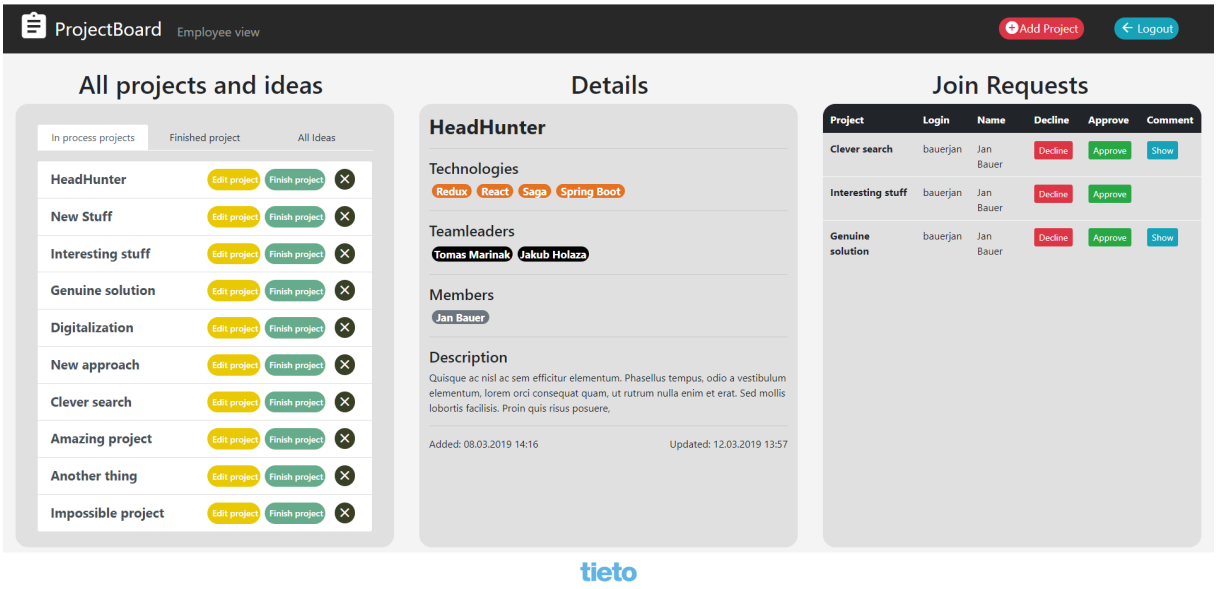
Obrázek 12: Employee page 1 – přehled



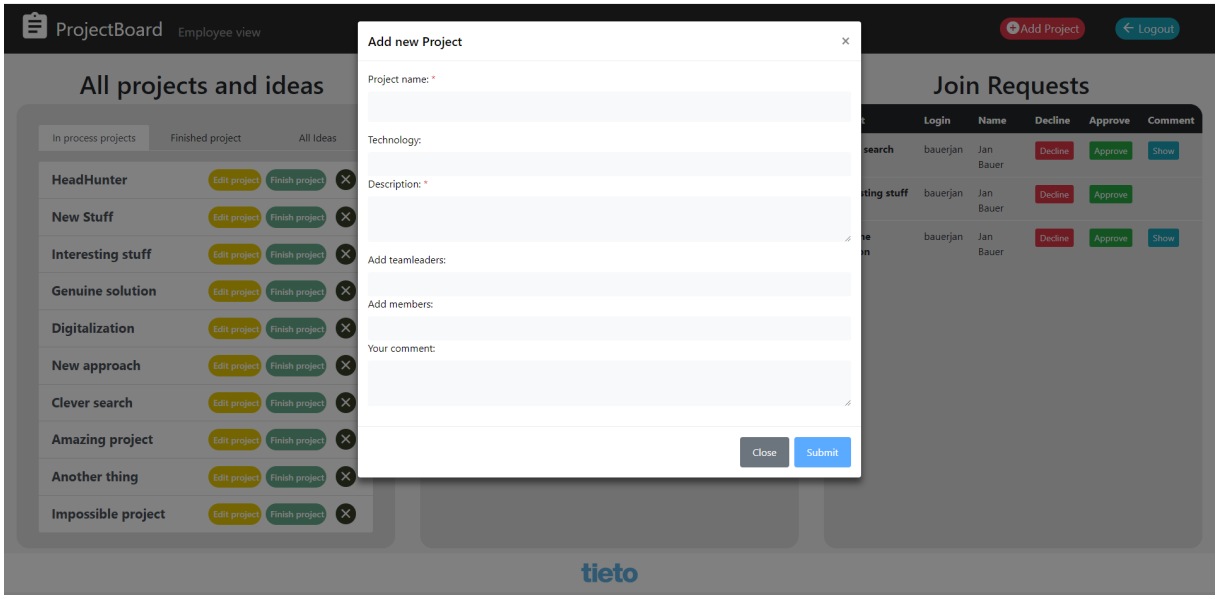
Obrázek 13: Employee page 2 – přidaní nápadu



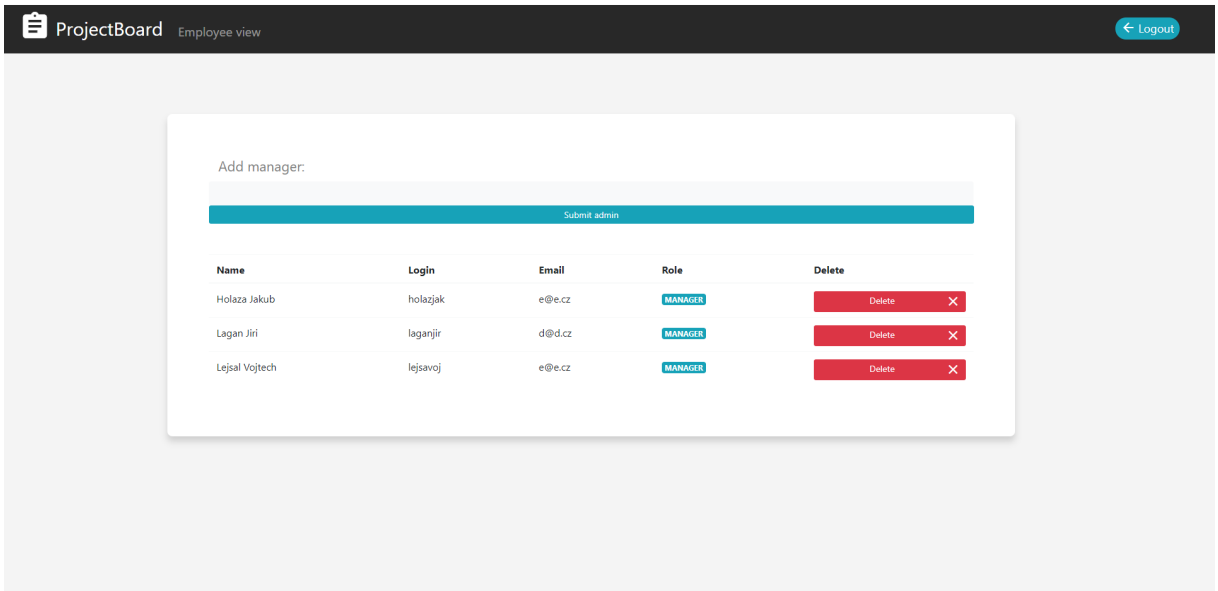
Obrázek 14: Employee page 3 – My desktop



Obrázek 15: Manager page 1 – přehled



Obrázek 16: Manager page 2 – přidání projektu



Obrázek 17: Admin page

6 Znalosti a dovednosti získané v průběhu studia uplatněné v průběhu odborné praxe

Na této praxi jsem z velké míry využil znalosti ze studia, hlavně obecně z objektově orientovaného přístupu k programování již z druhého semestru z předmětu Programování 2, dále z předmětu Programovací jazyky 1, které jsme pracovali s programovacím jazykem Java, v kterém teď hlavně pracuji. Pak jsem využil znalosti z předmětu Softwarového inženýrství k návrhu aplikací s pomocí jazyka UML, díky čemuž jsem si lépe dokázal ujasnit a uchopit, jak aplikace bude fungovat a jaké akce v ní budou probíhá, což bylo ještě rozšířeno v předmětu Vývoj informačních systému. Zjistil jsem, že na začátku vývoje aplikace je důležité si správně si definovat a zvážit požadavky na aplikaci a následně na to správně rozvrhnout architekturu aplikace, aby se předešlo budoucím chybám při samotné implementaci. V této závislosti mi hodně pomohla znalost SQL jazyka z předmětů Úvod do databázových systémů a Databázové a informační systémy, kde jsem se také naučil, jak navrhnout a posoudit relační datový model databáze a jak s touto databází skrze ORM pracovat a provádět operace s jejími daty tedy sestavit business logic. Potom mi také byly k užitku informace z předmětu Tvorba aplikací pro mobilní zařízení I, kde jsem se seznámil se základním vývojem pro web a ještě jsem využil znalosti z předmětu Uživatelská rozhraní, kde jsem zjistil jaké jsou pravidla pro návrh uživatelského rozhraní aplikací a jaké vizuální prvky by aplikace měla mít, aby byla přehledná a lépe se uživateli intuitivně používala.

7 Znalosti či dovednosti scházející studentovi v průběhu odborné praxe

Na univerzitě jsem dostal potřebný základ pro programování a dále jsem také z velké míry přivítal možnost si vybrat z dostatečně velkého množství volitelných předmětů, dle kterých jsem si mohl navolit to, co mě zajímá, a více si tak ujasnit, na jakou oblast bych se chtěl případně později orientovat, ale asi stejně nejde obsáhnout úplně vše. Nicméně na univerzitě bych nyní z mého hlediska popřípadě uvítal možná více předmětů zaměřených na vývoj aplikací v JavaScriptu, který disponuje velkým rozsahem knihoven pro využití k vývoji různého druhu aplikací, například v předmětu podobnému Skriptovací programovací jazyky, kde by byla probírána některá hojně používaná javascriptová knihovna nebo framework jako React nebo Angular, které se využívají v praxi. Poté by mohl být k dispozici například i volitelný předmět zabývající se No-SQL databázemi jako Mongo, Neo4j, apod., které ve firmě právě také používáme. Jinak oceňuji to, že se na univerzitě konají i přednášky přímo od firem nebo vývojářů z praxe, kdy je možno vidět, jaké technologie a přístupy se v daných oblastech informatiky používají.

8 Závěr

Na bakalářské praxi jsem se hlavně naučil pracovat s danými frameworky, technologiemi, moduly, balíčky a knihovnami třetích stran a také i pro mě novými vývojovými prostředími a jejich funkcemi, které se používají pro vývoj webových aplikací. Pro tyto účely jsem se také seznámil s build a dependecy manažerem Apache Maven pro přidávání závislostí na knihovny v backendové části a s Node.js npm build a balíčkovým manažerem pro přidávání modulů ve frotendové části. Dále jsem také zjistil, jaké je to pracovat v týmu z hlediska vývojové metodiky SCRUM, jak si rozdělit práci na projektu a prezentovat jeho postup, používat verzovací systém Git. Získal jsem i zkušenosti s nasazením aplikace na testovací server a nakonec se samotným testováním softwaru. Což znamená, že pokaždé když někdo z týmu dokončí svou zadanou aplikaci, tak je třeba aplikaci prověřit, kdy se víceméně celý tým kolegů sejde a probíhá testování a zapisují se případné chyby nebo připomínky ke korekci.

Tento čas strávený na této praxi pro mě byl velkým přínosem z hlediska praktických zkušeností a pohledu na obor informačních a výpočetních technologií celkově v příjemném a přátelském prostředí.

Literatura

[1] *Tieto* [online] [cit. 10. března 2019]
https://www.tieto.com/cz/about-us/our-company/

[2] *PostgreSQL* [online] [cit. 12. března 2019]
https://www.postgresql.org/

[3] *Spring Framework* [online] [cit. 12. března 2019]
https://spring.io/projects/spring-framework

[4] *Spring Boot* [online] [cit. 12. března 2019]
https://spring.io/projects/spring-boot

[5] *Exploring Micro-frameworks: Spring Boot* [online] [cit. 12. března 2019]
https://www.infoq.com/articles/microframeworks1-spring-boot

[6] *JPA, Hibernate* [online] [cit. 14. března 2019]
https://thoughts-on-java.org/difference-jpa-hibernate-eclipselink/

[7] *React* [online] [cit. 15. března 2019]
https://reactjs.org/

[8] *React JSX* [online] [cit. 15. března 2019]
https://reactjs.org/docs/introducing-jsx.html

[9] *Redux* [online] [cit. 15. března 2019]
https://redux.js.org/introduction/getting-started

[10] *Saga* [online] [cit. 15. března 2019]
https://redux-saga.js.org/

[11] *What is Redux-Saga?* [online] [cit. 15. března 2019]
https://engineering.universe.com/what-is-redux-saga-c1252fc2f4d1

[12] *Git* [online] [cit. 20. března 2019]
https://git-scm.com/

[13] *GitLab* [online] [cit. 20. března 2019]
https://about.gitlab.com/what-is-gitlab/

[14] *Bootstrap* [online] [cit. 21. března 2019]
https://getbootstrap.com/

[15] *Jetbrains* [online] [cit. 21. března 2019]
https://www.jetbrains.com/idea/

[16] *Java* [online] [cit. 3. dubna 2019]
https://www.java.com/en/download/faq/whatis-java.xml/

[17] *Typescript* [online] [cit. 3. dubna 2019]
https://www.typescriptlang.org/